

# Ranking Under Temporal Constraints

Lidan Wang  
Dept. of Computer Science  
University of Maryland  
College Park, MD  
lidan@cs.umd.edu

Donald Metzler  
Information Sciences Institute  
Univ. of Southern California  
Marina del Rey, CA  
metzler@isi.edu

Jimmy Lin  
The iSchool  
University of Maryland  
College Park, MD  
jimmylin@umd.edu

## ABSTRACT

This paper introduces the notion of temporally constrained ranked retrieval, which, given a query and a time constraint, produces the best possible ranked list within the specified time limit. Naturally, more time should translate into better results, but the ranking algorithm should always produce *some* results. This property is desirable from a number of perspectives: to cope with diverse users and information needs, as well as to better manage system load and variance in query execution times. We propose two temporally constrained ranking algorithms based on a class of probabilistic prediction models that can naturally incorporate efficiency constraints: one that makes *independent* feature selection decisions, and the other that makes *joint* feature selection decisions. Experiments on three different test collections show that both ranking algorithms are able to satisfy imposed time constraints, although the joint model outperforms the independent model in being able to deliver more effective results, especially under tight time constraints, due to its ability to capture feature dependencies.

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

**General Terms:** Algorithms, Performance

**Keywords:** efficiency, linear models, learning to rank

## 1. INTRODUCTION

Most existing ranking functions are rigidly defined and incapable of adapting to diverse retrieval scenarios. In this work, we are interested in developing ranking functions that can robustly adapt to different efficiency constraints that may arise in real-world retrieval applications. As we will show, our proposed *temporally constrained ranking functions* are capable of producing high quality results given a predetermined amount of time. Naturally, the quality of results improves as more computation time is allowed, but critically, the ranking function should produce *some* results

given an arbitrary time constraint. This idea is related to *anytime algorithms*, introduced in the mid-1980s by Dean and Boddy [11] in the context of time-dependent planning. The primary difference, however, is that anytime algorithms provide a solution at any arbitrary point, whereas our temporally constrained ranking functions require the time constrained to be specified *in advance*.

More formally, we define a temporally constrained ranking function as one that solves the following ranking task: given a user query  $q$  and a time constraint  $T(q)$ , the goal is to produce a top  $k$  ranking of documents from collection  $C$  that maximizes a metric of interest such as mean average precision, NDCG, etc. Why is the ability to impose dynamic temporal constraints on a ranking function important? We motivate this problem from three separate angles: users, information needs, and system architectures.

First, users are diverse and have different tolerances to query execution times. Some are impatient and want results as soon as possible, even if the results may be of lower quality. For other users, waiting a bit longer may be acceptable if it means better results. Profiling of users along these lines is possible with log analysis, for example, by noting how frequently users click on the browser stop button.

Second, information needs are diverse. For example, a search for a simple factoid should be handled immediately, and as long as the desired information appears in the top hit or near the top, spending additional time on ranking represents wasted effort. On the other hand, complex informational queries might benefit from additional processing that, for example, takes into account term proximity and complex document features.

Finally, temporally constrained ranking functions are interesting from a systems perspective. In real-world settings, high query throughput is usually achieved by service replication and spreading the query load across a large number of servers. System capacity is fixed or relatively static (i.e., there are only so many available servers in a datacenter). Unfortunately, query load is highly variable. In addition to daily cycles, a system might experience sudden spikes in usage (i.e., the *flash crowd* effect). How a system behaves under load depends on the exact architecture, but temporally constrained ranking functions can help the system adapt. For example, when query load is high, we might tighten the temporal constraints to maintain roughly the same query latency, at the cost of some reduction in quality—this might be preferable to forcing users to wait longer for results.

Temporally constrained ranking functions can also help manage variance in query execution times. For real-world

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CIKM'10, October 26–30, 2010, Toronto, Ontario, Canada.

Copyright 2010 ACM 978-1-4503-0099-5/10/10 ...\$10.00.

services, we are not only interested in the mean, i.e., how long *on average* a user needs to wait for results, but also outliers, i.e., how slow the bottom 5% of queries are. Controlling instances of the latter case is very important, since they may represent dissatisfied users who never come back. Finally, retrieval may comprise only one component in a larger system (e.g., for summarization, interactive browsing, etc.). Successful composition of services often requires some guarantees (e.g., service level agreements) on the running times of individual components.

It is true that we can address the above issues by devising multiple ranking algorithms that encode specific tradeoffs, and then dynamically select the appropriate algorithm at query time. The obvious downside, however, is duplicate development effort. Instead, it would be more desirable to have a single ranking algorithm that comes with an adjustable “knob” to set the desired query evaluation time.

This work has three primary contributions. First, we introduce the notion of temporally constrained ranking functions for information retrieval. Second, we propose two methods for automatically constructing and learning temporally constrained ranking functions. Finally, we introduce a new metric that accounts for retrieval effectiveness as a function of time, which provides an objective function in our learning-to-rank approach. Experimental results on multiple TREC test collections show that our proposed ranking functions are effective across a range of temporal constraints.

The remainder of the paper is organized as follows: We start with a discussion of related work. Section 3 describes temporally constrained linear models and two algorithms for constructing such models. Our methods are evaluated in Section 4, before discussing future work and concluding.

## 2. RELATED WORK

To our knowledge, no existing information retrieval model can satisfy arbitrary time constraints. A particular model, be it probabilistic [25], LM-based [24, 30], or machine learned [19, 4], induces a fixed computational cost that can be quantified in terms of the number of postings traversed, floating-point computations required, etc. It is not possible to force a ranking model to return lower-quality results early, nor is it possible to achieve higher-quality results if given more time. The closest related work is that of Wang et al. [29], who propose learning-to-rank methods that incorporate both effectiveness and efficiency metrics. However, the learned models merely select a more optimal operating point in the space of effectiveness/efficiency tradeoffs, but do not provide control on an individual query basis, unlike in this work. The work of Collins-Thomspon [10] is also relevant. The proposed approach uses convex programming to choose query expansion terms, incorporating estimates of retrieval cost. However, our work focuses on *ad hoc* retrieval, as opposed to query expansion.

It is important to distinguish our work from efficient query evaluation approaches. We build ranking functions that satisfy each query’s time requirement, whereas query evaluation strategies, such as early-termination techniques [7, 2, 28], simply improve the speed of existing ranking models, but do not provide an explicit mechanism for controlling the tradeoff between effectiveness and efficiency. The same critique applies to work on index pruning [8, 23], where the goal is to improve query evaluation speed by reducing the size of the index. While Shmueli-Scheuer et al. [27] consider

query evaluation under budgetary constraints, but similar to the aforementioned early-termination techniques, the work is concerned with efficient query evaluation, rather than building efficiency-minded ranking functions.

Caching of posting lists or search results provides another way to speed up query evaluation [3], but this is complementary to our proposed approach. The primary purpose of caching is to improve the efficiency of a *given* ranking function. In contrast, we learn ranking evaluations according to temporal constraints, where query evaluation and caching strategies are assumed to be given. Thus, our work is orthogonal to caching and can be integrated with any caching strategy by modifying the cost model.

Finally, our work is also related to machine-learned (linear) ranking functions [21, 13, 5, 19] and feature selection [14, 20]. However, none of the existing approaches address the problem of learning temporally constrained ranking functions. In fact, our proposed ranking functions generalize conventional machine-learned linear ranking functions. The two models converge to the same solution as the temporal constraint is relaxed.

## 3. RANKING WITH TIME CONSTRAINTS

This section formalizes the problem of temporally constrained ranking, where in addition to a query  $q$ , we specify a time constraint  $T(q)$  for the query. We first provide a brief overview of linear ranking functions, and introduce the concept of temporally constrained linear ranking functions. We then present novel formulations based on probabilistic prediction models for constructing temporally constrained functions. Finally, we discuss how to satisfy the time requirement and estimate model parameters.

### 3.1 Linear ranking functions

Linear ranking functions are a class of simple, yet effective ranking functions. Many widely used ranking models belong to this family of ranking functions [21, 17, 4, 13, 5]. Our use of this class of ranking models is motivated by their demonstrated effectiveness over publicly benchmarked retrieval tasks [1, 9].

A linear ranking function is characterized by a set of features  $F = f_1, \dots, f_N$  and the corresponding model parameters  $\Lambda = \lambda_1, \dots, \lambda_N$ . Each feature  $f_i$  is a function that maps a query-document pair  $(q, d)$  to a real value. The relevance of document  $d$  with respect to query  $q$  is computed as:

$$Score(q, d) = \sum_i \lambda_i f_i(q, d)$$

Hence, each document is scored by a weighted linear combination of the feature values computed over the document and query. Similar to previous work [4, 17], we assume that the model parameters  $\lambda_i$  take the following parametric form:

$$\lambda_i(q) = \sum_j w_j g_j(q)$$

where  $g_j$ ’s are meta-features defined over the query for each feature  $i$ , and  $w_j$ ’s are the free parameters. Hence,  $\lambda_i$  depends on  $q$  via  $g_j$  and  $w_j$ . Using this formulation, the ranking function can be re-written as:

$$Score(q, d) = \sum_j w_j \sum_i g_j(q) f_i(q, d)$$

Feature	Description
$f_{T,Dir}(q, D) = \log \left[ \frac{t f_{q,D} + \mu \frac{c f_q}{ C }}{ D  + \mu} \right]$	unigram $q$ in document $D$ (Dirichlet)
$f_{T,BM25}(q, D) = \frac{(k_1 + 1) t f_{q,D}}{k_1 \left( (1-b) + \frac{b D }{ D' } \right) + t f_{q,D}}$	unigram $q$ in document $D$ (BM25)
$f_{O,Dir,N}(q_j, q_{j+1}, D) = \log \left[ \frac{t f_{\#odN}(q_j, q_{j+1}, D) + \mu \frac{c f_{\#odN}(q_j, q_{j+1})}{ C }}{ D  + \mu} \right]$	ordered window “ $q_j q_{j+1}$ ” (span= $N$ ) in $D$ (Dirichlet)
$f_{O,BM25,N}(q_j, q_{j+1}, D) = \frac{(k_1 + 1) t f_{\#odN}(q_j, q_{j+1}, D)}{k_1 \left( (1-b) + \frac{b D }{ D' } \right) + t f_{\#odN}(q_j, q_{j+1}, D)}$	ordered window “ $q_j q_{j+1}$ ” (span= $N$ ) in $D$ (BM25)
$f_{U,Dir,N'}(q_j, q_{j+1}, D) = \log \left[ \frac{t f_{\#unN'}(q_j, q_{j+1}, D) + \mu \frac{c f_{\#unN'}(q_j, q_{j+1})}{ C }}{ D  + \mu} \right]$	unordered window “ $q_j q_{j+1}$ ” (span= $N'$ ) in $D$ (Dirichlet)
$f_{U,BM25,N'}(q_j, q_{j+1}, D) = \frac{(k_1 + 1) t f_{\#unN'}(q_j, q_{j+1}, D)}{k_1 \left( (1-b) + \frac{b D }{ D' } \right) + t f_{\#unN'}(q_j, q_{j+1}, D)}$	unordered window “ $q_j q_{j+1}$ ” (span= $N'$ ) in $D$ (BM25)

**Table 1: Features used in our linear ranking functions.** Here,  $t f_{e,D}$  is the number of times concept  $e$  matches in document  $D$ ,  $c f_e$  is the number of times concept  $e$  matches in the entire collection,  $|D|$  is the length of document  $D$ ,  $|D'|$  is the average document length in the collection, and  $|C|$  is the total length of the collection.  $N$  and  $N'$  are the window sizes for ordered and unordered phrases, respectively, where  $N \in \{1, 2, 4\}$  and  $N' \in \{2, 4, 8\}$ . Finally,  $\mu$  is a Dirichlet feature scoring function hyperparameter,  $k_1$  and  $b$  are BM25 feature scoring function parameters.

We now describe the features  $f_i$  and  $g_j$  that we consider in this work. Both term-based features [24, 30] and term proximity features [6, 21] have been widely used in such models, and have been shown to be especially successful when used in combination [20]. We take a similar feature-oriented approach and use both term features and term proximity features as part of our feature pool. Table 1 provides a summary of the query-document features  $f_i$  considered in this work. Among them, we use two different unigram term features, each using a different feature scoring function (BM25 or Dirichlet). We use a set of term proximity features defined over bigrams within the query, where each is computed from a BM25 or Dirichlet scoring function, for a specific window type (ordered/unordered) and window length. These are similar to features previously explored [20].

For the query-dependent meta-features  $g_j$ , we use features similar to those described by Bendersky et al. [4], including collection-based (collection frequency and document frequency), features from external sources (English Wikipedia and a large Web collection), and a constant feature. The meta-features are summarized in Table 2.

Note that as a result of this query-dependent weighting [4, 17], features  $f_i$  that are defined on the same unigram/bigram (i.e., query concept) in the query will have the same feature weight value  $\lambda_i$ . This property will be used for developing efficient algorithms for creating temporally constrained ranking functions in Section 3.3. Most of the computational cost associated with such ranking functions comes from the cost incurred from evaluating the features  $f_i$ . Unlike the features  $f_i$ , we assume that there is negligible cost associated with computing the meta-features  $g_j$ . Typically, in an operational setting retrieval engines would have access to large-scale, low-latency distributed caches for these types of global statistics.

### 3.2 Constrained linear ranking functions

In a temporally constrained setting, using all features in the linear ranking function may exceed the time requirement. Instead, for each query, we need to alter the efficiency characteristics of the ranking function by using only a subset

Feature	Description
$g_1^t(q)$	# times $q$ occurs in the collection
$g_2^t(q)$	# documents $q$ occurs in the collection
$g_3^t(q)$	# times $q$ occurs in ClueWeb
$g_4^t(q)$	# times $q$ occurs in a Wikipedia title
$g_5^t(q)$	1 (constant feature)
$g_1^b(q_j, q_{j+1})$	# times bigram occurs in the collection
$g_2^b(q_j, q_{j+1})$	# documents bigram occurs in the collection
$g_3^b(q_j, q_{j+1})$	# times bigram occurs in ClueWeb
$g_4^b(q_j, q_{j+1})$	# times bigram occurs in a Wikipedia title
$g_5^b(q_j, q_{j+1})$	1 (constant feature)

**Table 2: Meta-features in our model.**

of the features to meet its time requirement. We call the resulting linear ranking function a *temporally constrained* linear ranking function and it has the following general form:

$$Score(q, d) = \sum_{f_i(q, \cdot): S_i=1} \lambda_i f_i(q, d)$$

$$s.t. \quad \sum_{f_i(q, \cdot): S_i=1} C(f_i(q, \cdot)) \leq T(q)$$

where  $C(f_i(q, \cdot))$  denotes the computational cost of evaluating feature  $f_i$  for  $q$  over the document collection,  $T(q)$  is the time requirement for query  $q$ ,  $S_i$  is a binary value denoting whether or not  $f_i(q, \cdot)$  is used for  $q$ , and  $\lambda_i$ 's are the linear feature weights, parameterized by meta-features as described in the previous section.

To instantiate a model, we must address the following: 1) how to best select the subset of query dependent features to construct the corresponding temporally constrained ranking function for each query  $q$ ; 2) how to define the cost function  $C(f_i(q, \cdot))$  for the linear features  $f_i(q, \cdot)$  such that the response time of the resulting ranking function will not exceed the time requirement; 3) how to determine the free parameters (i.e., meta-feature weights  $w_j$ ) for the temporally constrained ranking function. We described our proposed solution to these issues in the following sections.

### 3.3 Prediction models

This section introduces two methods for creating temporally constrained ranking functions. Both are based on a class of probabilistic prediction models that can naturally impose time constraints on query execution times. The first method, which we call “Indep”, makes independent decisions when selecting which features  $f_i$  should be evaluated. The second method, which we call “Joint”, goes an extra step by accounting for query-level feature redundancy. We note that in contrast to previously proposed feature selection methods for ranking [14, 20], our problem is unique in that the feature selection is driven by the time constraint for each query, not purely by effectiveness.

In both methods, the decision on what features to use largely depends on the feature weights  $\lambda_i(q)$ . Features with large weights are more likely to be selected when facing a time constraint because they have more impact on the final ranking. In particular, in the “Indep” model, the likelihood of a feature  $f_i(q, \cdot)$  being selected is directly proportional to its query dependent feature weight  $\lambda_i(q)$ . In the “Joint” model, as we will show, the selection depends both on the feature weight and the feature’s redundancy relationship with respect to other features.

#### 3.3.1 Independent Prediction Model

The independent prediction model is based on a logistic regression model. It aims to directly estimate the likelihood that a given feature  $f_i$  should be included in the ranking function. Using this model, the probability of selecting feature  $f_i$  (i.e.,  $S_i = 1$ ) is computed as:

$$P(S_i|q) = \frac{\exp(\lambda_i(q))}{1 + \exp(\lambda_i(q))} = \frac{\exp\left(\sum_j w_j g_j(q, i)\right)}{1 + \exp\left(\sum_j w_j g_j(q, i)\right)} \quad (1)$$

where the  $g_j$ ’s are the meta-features used in our linear ranking function and the  $w_j$ ’s are the corresponding model parameters. When defined this way, the likelihood of choosing feature  $i$  is monotonic with respect to  $\lambda_i(q)$ , the query-dependent weight from our linear ranking function. As mentioned earlier, this is intuitively appealing, as it is desirable to choose features with large weights, since they are more likely to have greater impact on the ranking.

Under this independent model, the joint probability of selecting features is simply the product of individual selection likelihoods. Given a time constraint  $T(q)$  for query  $q$ , we need to infer the most likely selections over all features and construct the corresponding constrained ranking function  $R(q)$ , such that the time cost of  $R(q)$  will be within  $T(q)$ . While many existing methods are available for general inference for prediction models [15], such methods are unconstrained. Instead, inference under a time constraint  $T(q)$  for a given query can be shown to be equivalent to the following optimization problem, after doing a few simple logarithmic operations on Equation 1:

$$\begin{aligned} \hat{S} &= \arg \max_S \sum_i S_i \lambda_i(q) & (2) \\ \text{s.t.} \quad & \sum_i S_i C(f_i(q, \cdot)) \leq T(q), \quad S_i \in \{0, 1\} \end{aligned}$$

where  $S_i$  is a binary variable indicating whether feature  $f_i$  and its corresponding feature weight  $\lambda_i$  will be used in the ranking function or not,  $C(f_i(q, \cdot))$  is the cost of evaluating

---

#### Algorithm 1: Independent Ranking

---

**Input:** Query execution time requirement  $T(q)$ ;  
 ranking features  $f_i(q, \cdot)$ ; meta-features  $g_j(q)$   
 and meta-features weights  $w_j$   
**Output:** Temporal constrained ranking function  $R(q)$   
 Compute feature weights:  $\lambda_i(q) = \sum_j w_j g_j(q)$ ;  
 Compute feature profit density:  $p_i = \frac{\lambda_i(q)}{C(f_i)}$ ;  
 Queue  $\mathcal{F}$ : features sorted by decreasing profit density;  
 Initialize  $R(q) = \{\}$ ;  
 Initialize  $totalCost = 0$ ;  
**while**  $size(\mathcal{F}) > 0$  **do**  
   Remove  $f_i$  from head of  $\mathcal{F}$  ;  
   **if**  $totalCost + cost(f_i) < T(q)$  **then**  
     add  $\langle f_i, \lambda_i \rangle$  to ranking function  $R(q)$ ;  
      $totalCost = totalCost + cost(f_i)$ ;  
**end**  
**return**  $R(q)$ ;

---

$f_i$  for  $q$  over the collection, and  $T(q)$  is the time constraint. The goal of the optimization is to find an optimal set of features that maximizes the objective and satisfies the time constraint  $T(q)$ . Similar approaches for casting inference as an optimization task have also been used in the NLP task of semantic role labeling [26]. This formulation has the advantage that it enables us to impose arbitrary linear constraints (such as temporal constraints) on the model outputs (i.e., the temporally constrained ranking function).

Solving the stated optimization problem in Equation 2 is done at runtime, so for each query, the complexity cannot be very high. Nonetheless, the Indep model allows for an extremely efficient inference from which the temporally constrained ranking function is constructed. Note that our query cost model excludes the inference cost because, as we will show, the running time complexity for constructing ranking functions is negligible for reasonably sized queries.

The process of inferring the temporal ranking function for the Indep model is presented in Algorithm 1. It should be easy to see that for the Indep prediction model, the optimization problem corresponds to the classic *knapsack problem*. Here, features are “items” that we are trying to fit into the knapsack. The value of each corresponds to the feature weight  $\lambda_i(q)$ , the cost corresponds to  $C(f_i(q, \cdot))$ , which will be explained in detail in Section 3.4, and the knapsack capacity is  $T(q)$ . We first compute profit density for each feature  $f_i$ , which is defined by the ratio between its value and cost (i.e.,  $\frac{\lambda_i(q)}{C(f_i)}$ ). We then add features according to this density (largest first), until we can no longer add any more features without overshooting our time constraint, or until when we run out of features to use.

Since the number of features is linear in  $|q|$ , the length of the query, the time complexity for this process is  $O(|q| \log |q|)$ , which accounts for sorting and adding features into the constrained ranking function. This is trivial compared to the time taken for query evaluation.

#### 3.3.2 Joint Prediction Model

The independent prediction model assumes that feature selection decisions are made independently of each other. While this assumption is reasonable for small feature sets, it may not hold as well for larger feature sets. Therefore,

we would like to model relationships between features and eliminate the computation of redundant features that do not add very much to the ranking function in terms of relevance, yet result in increased query execution times.

We define redundancy only for features defined over the same query concept (i.e., unigram or bigram). The intuition is that under a time constraint, we would like to use features that provide maximal coverage over all query concepts, rather than repeatedly using features for the same query concept. Hence, for a pair of features  $f_i$  and  $f_j$  that share a common query concept (i.e.,  $\lambda_i(q) = \lambda_j(q)$ ), the redundancy is defined as follows:

$$r(f_i, f_j) = \begin{cases} 1 & \text{if } \lambda_i < \alpha \\ 0 & \text{otherwise} \end{cases}$$

where  $\alpha$  is a weight threshold. This definition says that  $f_i$  and  $f_j$  are redundant if the feature weight does not exceed threshold  $\alpha$ . This definition allows us to impose redundancy penalty on features defined over less important query concepts, as determined by the feature weight value.

We propose using an undirected probabilistic graphical model [12] to capture the feature importance and redundancy relationship between features when making the selection under a time constraint. An undirected graphical model is a probabilistic model defined over an undirected graph, which encodes the conditional independence assumptions amongst random variables, corresponding to the nodes in the graph. Here, we make use of the Ising model (a.k.a. Boltzmann machine) [16] to represent individual features and their pairwise redundancy relationships. Under this model, the joint probability over all feature decisions is:

$$P(S_1, \dots, S_k) = \frac{1}{Z} \exp \left( \sum_i \left( \sum_j w_j g_j(q, i) S_i + \sum_{i' \in N(i)} \beta \cdot r(f_i, f_{i'}) S_i S_{i'} \right) \right) \quad (3)$$

where  $N(i)$  are the features that share an edge with feature  $i$  in the graph (i.e., features in  $N(i)$  are defined over same query concept as  $f_i$ ),  $r(f_i, f_{i'})$  is the redundancy feature defined over the pair  $(f_i, f_{i'})$  and  $\beta$  is the model parameter associated with redundancy feature  $r$ .  $S_i$ ,  $w_j$ , and  $g_j$  are as defined earlier.

Similar to the Indep model, after performing simple logarithmic operations on Equation 3, the probability of selecting features  $f_1, \dots, f_k$  under a time constraint can be stated by the following optimization problem:

$$\begin{aligned} \hat{S} &= \arg \max_S \sum_i \lambda_i(q) S_i + \sum_i \sum_{i' \in N(i)} \beta \cdot r(f_i, f_{i'}) S_i S_{i'} \\ \text{s.t. } &\sum_i S_i C(f_i(q, \cdot)) \leq T(q), \quad S_i \in \{0, 1\} \end{aligned}$$

The goal of the optimization is to find a set of features that reach an optimal balance between feature effectiveness and feature redundancy, while satisfying the time constraint  $T(q)$ . Note that when  $\beta = 0$  this problem reduces to the independent case.

The optimization problem is an integer linear program [22], and several practical solutions exist. Most commonly, they

---

### Algorithm 2: Joint Ranking

---

**Input:** Query execution time requirement  $T(q)$ ;  
ranking features  $f_i(q, \cdot)$ ; meta-features  $g_j(q)$   
and meta-features weights  $w_j$

**Output:** Temporal constrained ranking function  $R(q)$

Compute feature weights:  $\lambda_i(q) = \sum_j w_j g_j(q)$ ;

Compute feature profit density:  $p_i = \frac{\lambda_i(q)}{C(f_i)}$ ;

Queue  $\mathcal{F}_1$ : features sorted by decreasing profit density;

Initialize queue  $\mathcal{F}_2 = \{\}$ ;

Initialize  $R(q) = \{\}$ ;

Initialize  $totalCost = 0$ ;

For each concept  $e$ , let  $G_e =$  set of features defined on  $e$ ;

Let  $\lambda_e$  denote the weight of features in  $G_e$ ;

Let  $covered[e] = \text{false}$  for all  $e$ ;

**while**  $size(\mathcal{F}_1) > 0$  or  $size(\mathcal{F}_2) > 0$  **do**

Remove  $f_i$  that has max  $p_i$  from head of  $\mathcal{F}_1$  or  $\mathcal{F}_2$ ;

**if**  $totalCost + cost(f_i) < T(q)$  **then**

add  $\langle f_i, \lambda_i \rangle$  to ranking function  $R(q)$ ;

$totalCost = totalCost + cost(f_i)$ ;

Denote concept covered by  $f_i$  by  $e'$ ;

**if** ( $covered[e']$  is false and  $\lambda_{e'} < \alpha$ ) **then**

$\lambda_{e'} = \lambda_{e'} - \beta$ ;

$G_{e'} = G_{e'} \setminus f_i$ ; move  $G_{e'}$  from  $\mathcal{F}_1$  to end of  $\mathcal{F}_2$ ;

$covered[e'] = \text{true}$ ;

**end**

**return**  $R(q)$ ;

---

iteratively make decisions based on a score value for each feature (similar to the knapsack problem). However, the standard solutions require continually re-sorting the features after a feature is added and the values of its neighbor features are re-computed to account for redundancy. Our problem has the nice property that all features defined over the same query concept share a common feature weight value, and their redundancy features are binary valued. This property enables us to derive an efficient solution to the optimization problem (Algorithm 2) that has the same running time complexity as Algorithm 1. Experiments in Section 4 show that the Joint model is significantly more effective than the Indep model.

Similar to the independent case, we add features according to their profit densities (largest first). When a feature  $f_i$  is added, if its feature weight does not exceed  $\alpha$ , the selection likelihoods of its remaining neighboring features are recomputed to account for the feature redundancy penalty (if not already done). In this case, feature ordering needs to be adjusted to account for new weights. But because features defined on the same query concept share the same weight value, there is no need to resort them after they receive the same penalty  $\beta$ . To make sure they are properly placed with respect to the other features, we maintain two queues:  $\mathcal{F}_1$  contains non-penalized features as sorted originally, and  $\mathcal{F}_2$  contains the penalized features, which are always sorted as explained earlier. This way, there is no need to repeatedly perform re-ordering operations after new weights are computed. We select the feature with the best profit density between  $\mathcal{F}_1$  and  $\mathcal{F}_2$  as the next feature for the temporally constrained ranking function during each iteration.

The time complexity for this process is  $O(|q| \log |q|)$ , accounting for the initial sorting of features and for creating

the ranking function. In practice, this cost is trivial for reasonably sized queries.

### 3.4 Temporal constraint enforcement

In this section, we describe how we define  $C(f_i(q, \cdot))$ , the cost for evaluating feature  $f_i(q, \cdot)$  for  $q$  over the document collection. More specifically, we are interested in characterizing the efficiency of a linear ranking function as a result of using a particular subset of features. Efficiency can be interpreted in two ways: 1) in an absolute sense, in terms of the total amount of time necessary to compute the ranked list, or 2) in a relative sense, where given a baseline ranking model, we measure how much more (or less) efficient our proposed method is compared to the baseline. In our work, we adopt the second interpretation because it factors out differences in hardware.

Thus, we model the query execution time relative to some baseline ranking function. This type of cost model requires us to capture the costs associated with evaluating different features  $f_i$  within the linear ranking function. The cost for a given linear ranking function  $R(q)$  is computed as the sum of its individual feature costs:

$$C(R(q)) = \sum_{i=1}^N C(f_i(q, \cdot))$$

We estimate  $C(f_i(q, \cdot))$  as the sum of document frequencies  $DF(t)$  of each term  $t$  required to compute  $f_i$ . That is,

$$C(f_i(q, \cdot)) = \sum_{t \in f_i} DF(t)$$

For features defined over unigrams, this sum only has a single component, while for features defined over bigrams, it has two components. Intuitively, this analytical model captures the fact that evaluating more features and evaluating each feature over longer postings lists (i.e., large DF values) will both result in greater time complexity.

Given a baseline ranking function, we want to construct a temporally constrained ranking function  $R(q)$  such that the actual execution time of the model is within a multiple  $k$  of the baseline model’s query execution time  $R_b(q)$ . This can be expressed as the following constraint:

$$C(R(q)) \leq k C(R_b(q))$$

where the cost is computed as described above. Our algorithm for constructing  $R(q)$  will enforce this constraint. As will be shown in experiments, this very simple cost model works surprisingly well for ensuring the actual time  $T(R(q))$  of the model meets the actual time constraint:

$$T(R(q)) \leq k T(R_b(q))$$

where  $k T(R_b(q))$  denotes the time requirement for  $q$ , as a multiple of baseline time  $T(R_b(q))$ .

### 3.5 Parameter Estimation

It should be clear that a new optimization metric must be defined here, because commonly-used effectiveness metrics (MAP, P20, etc.) do not directly account for how the effectiveness of models varies across a range of time constraints. To evaluate the performance of a temporally constrained ranking function, the metric must account for the expected effectiveness over different time budgets. More formally, we

	Wt10g	Gov2	Clue
topics	451-550	701-850	1-50
# docs	1,692,096	25,205,179	50,220,423
avg qlen (title)	2.50	2.96	1.88
avg qlen (desc.)	6.08	5.90	5.88

**Table 3: Number of docs, topics, and average query lengths of test collections used in our experiments.**

define the expected effectiveness  $\mathcal{E}_q$  for a query  $q$  as:

$$\mathcal{E}_q = \sum_t e_q(t)P(t)$$

Where  $e_q(t)$  denotes the effectiveness achieved by the constrained ranking function associated with time constraint  $t$ , and  $P(t)$  represents the likelihood of  $t$  being assigned as the time constraint for the query. Any commonly-used effectiveness metric (such as average precision, P20, etc.) can be used in the effectiveness measure  $e_q(t)$  at each single time point  $t$ . In our experiments section, we report results from using average precision and P20.

For simplicity, we assume all  $t$  values are equally likely (i.e., uniform distribution for  $P(t)$ ). However, we can assign various other distributions (i.e., Gaussian, exponential, etc.) to model more complex distributions for time constraints.

Taking the mean of  $\mathcal{E}$  over a set of  $n$  queries gives us the mean expected effectiveness:

$$\mathcal{M}_E = \frac{1}{n} \sum_q \mathcal{E}_q$$

which represents the overall performance of the ranking functions across different time constraints and queries.

The set of free parameters we need to learn offline are the meta-feature weights  $w_j$  and the redundancy feature weight  $\beta$  and threshold  $\alpha$  (Joint model only). We note that the values of these parameters versus the objective metric  $\mathcal{M}_E$  is not smooth and there are multiple local maxima, so the standard gradient-search based methods cannot be applied. Instead, we employ a simple line search algorithm for optimizing the various model parameters by directly optimizing the mean expected effectiveness  $\mathcal{M}_E$  on the training set. This approach has been used by several earlier works for estimating parameters in ranking functions to directly optimize objective metrics [4, 21]. It iteratively optimizes the metric by performing a series of one-dimensional line searches. At each iteration, it searches for an optimal value for a parameter while holding all other parameters fixed. This iterative process continues until the improvement in the objective metric drops below a threshold. More details of this parameter estimation can be found in [22].

## 4. EXPERIMENTS

We report experimental evaluation of our proposed temporally constrained ranking models on three TREC web test collections: Wt10g, Gov2, and Clue (first English segment of ClueWeb09). Details of these test collections are provided in Table 3. The title and description portions of the corresponding TREC topics were used as queries, split equally into a training and test set. All parameter tuning was performed on the training set, and all of the results reported are from applying the learned parameters to the test set.

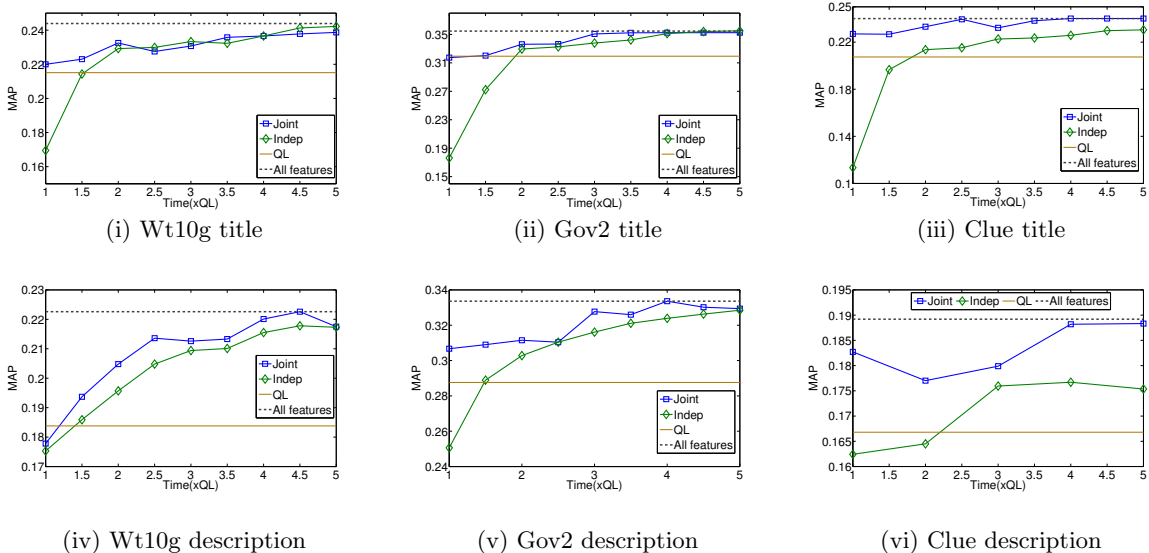


Figure 1: MAP versus time for the Indep and Joint models on title and description queries in the test sets of Wt10g, Gov2 and Clue. Effectiveness of baseline QL is plotted as the lower solid line; “All features” upper-bound is plotted as the upper dotted line.

	Wt10g	Gov2	Clue
Title	0.18s	2.2s	4.2s
Description	0.48s	7.7s	20.5s

Table 5: Avg. query execution time of baseline QL for title and desc. queries of Wt10g, Gov2, Clue.

Our experiments compare average precision (AP), precision at 20 (P20), and mean expected effectiveness  $\mathcal{M}_E$  of various models, including a baseline query-likelihood (QL) model, a baseline sequential dependence (SD) model [21], the Indep model, and the Joint model. In addition, we constructed an upper-bound model, called “All features”, which is a standard learning-to-rank model (i.e., optimized for effectiveness only with no temporal constraints) trained over the entire feature set. To remain consistent with our proposed models, the “All features” model is constructed in the same manner as the temporally constrained models, except the temporal constraint is set to infinity. The Wilcoxon signed rank test with  $p < 0.05$  was used to test for statistically significant differences between the methods.

We implemented our framework on top of Ivory, a newly developed open-source web-scale information retrieval engine [18]. All experiments were run on a SunFire X4100, with two Dual Core AMD Opteron Processor 285 at 2.6GHz and 16GB RAM (although all experiments used only a single thread). Recall that we varied the time constraint for each query  $q$  to be a multiple  $k$  of the query execution time of baseline QL, denoted by  $k \cdot T_{QL}(q)$ . The average query execution times of baseline QL for various data collections and queries are shown in Table 5 for reference.

#### 4.1 Effectiveness vs Time Constraints

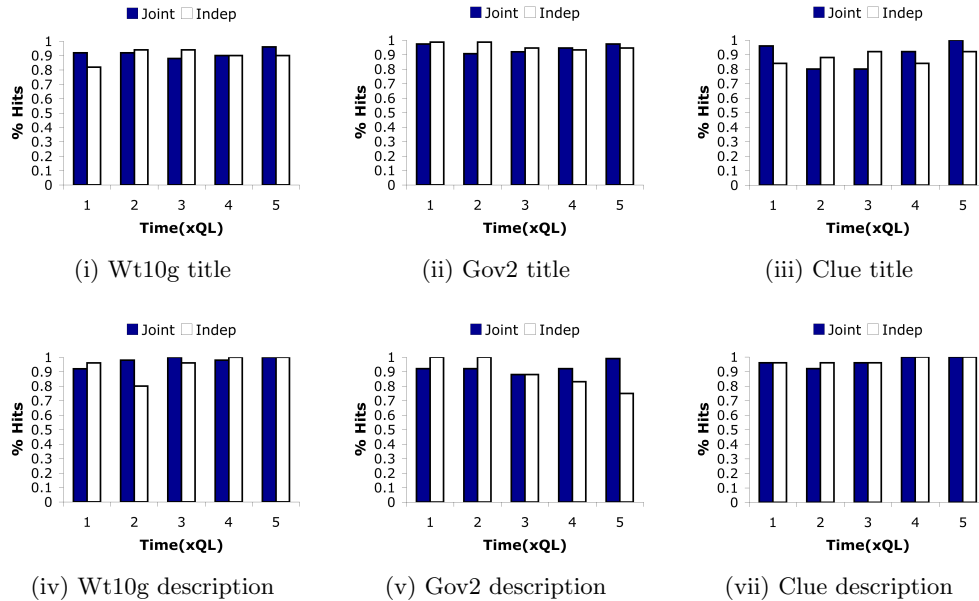
Figure 1 compares the MAP of our proposed temporally constrained ranking algorithms as a function of time constraints, from  $1 \cdot T_{QL}(q)$  to  $5 \cdot T_{QL}(q)$  in increments of  $0.5 \cdot T_{QL}(q)$  for each query (title, top row; description, bottom

row). In each graph, the effectiveness of the baseline QL model is plotted as the lower solid line, and the “All features” upper-bound is plotted as the upper dotted line.

While in general both the Indep and Joint models produce more effective results than baseline QL when given more time, the Joint model consistently achieves equal or higher MAP than the Indep model across all time constraints for both title and description queries. It also approaches the upper bound more rapidly as the time constraint is relaxed. This shows that the Joint model is superior to the Indep model across a wide range of retrieval scenarios.

Another interesting aspect is that the effectiveness difference between the Indep and Joint models is largest under tight temporal constraints, while the two models tend to perform similarly when the temporal constraint is relaxed. This can be explained by the fact that eliminating redundant features is more critical under strict temporal constraints—in such a case, using a diverse set of features is preferred. The effect of redundant features diminishes as the time constraint increases, since there is sufficient time to evaluate more features (redundant or otherwise). Feature redundancy also explains why the Indep model is different with respect to Joint and QL for very strict temporal constraints: it repeatedly selects features based on profit density alone. Thus, the time constraint will likely be violated before (at least) one feature from each facet in the query can be selected. Note this is not a problem with baseline QL, which only uses one feature for each term so there are no redundant features. In other words, given a rich feature set, under a strict time constraint, effectiveness is more critically dependent on selecting both high quality and non-redundant features.

In general, MAP increases as the time constraint is relaxed, for both the Joint and Indep models. Once again, this makes sense since the models are able to benefit from more features. However, we encounter a point of diminishing returns around  $4 \cdot T_{QL}$ , which suggests that using a very large number of features, as done by previous effectiveness-



**Figure 2: Bar chart showing the fraction of query evaluation times that satisfy the imposed time constraint for title and description queries from Wt10g, Gov2, and Clue.**

centric ranking models, may not be very desirable in our formulation of temporally constrained ranking, at least on these datasets. In a learning-to-rank scenario that only takes effectiveness into account (which might roughly correspond to the right edges of our graph), we feel that the marginal gains in effectiveness that comes with evaluating more features (thus taking more time) trades off too much efficiency for a small effectiveness gain. This speaks to the advantage of our temporally constrained ranking functions, which allows the tradeoff between effectiveness and efficiency to be explicitly considered.

## 4.2 Satisfying Time Constraints

Recall that our constrained ranking functions use an analytical cost model based on the sum of document frequencies, which are highly correlated with actual query evaluation times. However, whether the time constraints are *actually* met on the test data is an empirical question, since there are no theoretical guarantees. As described in Section 3.3, the time for constructing ranking functions is negligible compared to retrieval time for reasonably sized queries, thus it is not included in the query evaluation times.

Figure 2 illustrates how well, in reality, the temporally constrained ranking functions satisfy query-specific time constraints on the test data. Each set of bars represents a particular time constraint, defined as before in terms of multiples of the baseline QL query evaluation time. The height of each bar shows the fraction of queries that satisfy the required time constraint. For each time constraint  $t$ , the temporally constrained ranking function is said to have satisfied the constraint if the actual query execution time is less than or equal to  $t$ . We see that the “hit rates” (i.e., meeting the specified time constraint) for both ranking algorithms are well above 80% for almost all test collections, while most of the values are above 90%. This suggests our simple sum-of-*dfs* cost model, which is used by the algorithm to guarantee the time constraints, works reasonably well in practice.

What is not shown on the graphs is that the same queries failed to meet multiple time constraints. For instance, the query “maps of United States” missed all of the time constraints for the Clue collection. We found the queries that often missed their targets usually contain many frequent terms and/or are longer than average. This suggests that our analytical model underestimates the cost of these ‘outlier’ queries. As part of future work, we would like to explore ways to improve our analytical model to better handle such queries to minimize the number of queries that miss the specified time constraint.

Finally, a query is classified as a “miss” if its response time is greater than the required time  $t$ . However, if the time exceeds  $t$  by only a small amount, it may still be tolerable in practice. For instance, by allowing the response time to exceed  $t$  by a small amount (e.g., 5%), we observed that our models achieve 100% hit rates in many cases. Further details are omitted due to space constraints.

## 4.3 Expected Effectiveness Across Constraints

To investigate the average effectiveness of our proposed temporally constrained ranking algorithms across different time points, we compared their mean expected effectiveness (according to MAP and P20) in Table 4 for title and description queries across all three test collections. Results from the QL baseline are also reported. In terms of averaged effectiveness computed using MAP, the Joint model consistently and significantly outperforms both Indep and QL. Specifically, for title queries, the Joint model attains 7.5%, 6.8%, and 13.3% improvements over QL on Wt10g, Gov2, and Clue, respectively. The Joint model achieves 2.7%, 7.7%, and 13.1% gains over the Indep model on the same collections, respectively. Similar improvements are observed under averaged effectiveness computed using P20. We also note that the improvements of Joint over QL and Joint over Indep are statistically significant in two out of three collections, under both of the effectiveness metrics. The Indep

Title Queries

	Wt10g			Gov2			Clue		
	$M_E$ (MAP)	$M_E$ (P20)	$T_{98}$	$M_E$ (MAP)	$M_E$ (P20)	$T_{98}$	$M_E$ (MAP)	$M_E$ (P20)	$T_{98}$
QL	21.51	32.40	–	31.94	50.93	–	20.74	37.25	–
Indep	22.54 (+4.7)	32.77 (+1.1)	4.5	31.67 (–0.8)	50.37 (–1.1)	4	20.78 (+0.2)	35.01 (–6.0)	–
Joint	<b>23.14*</b> (+7.5/+2.7)	<b>33.22</b> (+2.5/1.4)	4.5	<b>34.12*</b> (+6.8/+7.7)	<b>54.76*</b> (+7.5/8.7)	3	<b>23.51†</b> (+13.3/+13.1)	<b>40.54*</b> (+8.8/15.7)	2.5

Description Queries

	Wt10g			Gov2			Clue		
	$M_E$ (MAP)	$M_E$ (P20)	$T_{98}$	$M_E$ (MAP)	$M_E$ (P20)	$T_{98}$	$M_E$ (MAP)	$M_E$ (P20)	$T_{98}$
QL	18.38	28.60	–	28.76	49.47	–	16.68	29.90	–
Indep	20.35* (+10.7)	31.18* (+9.0)	–	30.76* (+7.0)	51.25 (+3.6)	5	16.93 (+1.5)	34.69 (+16.0)	–
Joint	<b>20.84*</b> (+13.4/+2.4)	<b>31.45*</b> (+10.0/+0.8)	4	<b>32.05*</b> (+11.4/+4.2)	<b>52.82*</b> (+6.8/+3.1)	3	<b>18.18</b> (+9.0/+7.4)	<b>37.05</b> (+24.0/+6.8)	4

Table 4: Mean average effectiveness in terms of averaged MAP and averaged P20 from times  $1 \cdot T_{QL}$  to  $5 \cdot T_{QL}$  for title and description queries in Wt10g, Gov2, and Clue. Bolded values denote best performance obtained for test queries in each dataset. The \* and † symbols represent statistically significant differences with respect to QL and Indep, respectively. Percentage improvement shown in parentheses: over QL for Indep, and over QL/Indep for Joint.

model improves over QL in two out of three collections, under both effectiveness metrics, although the improvements are not statistically significant for the title queries.

For the description queries, both the Indep and Joint models achieve significant improvements over QL across all collections, while the Joint model also outperforms the Indep model in all three collections. The improvements of the Indep model over QL in terms of average effectiveness using MAP are 10.7%, 7.0%, and 1.5%, respectively on the three collections. In two of these three cases the improvements of Indep over QL are statistically significant. Similar results are obtained under averaged effectiveness computed from P20. Compared to title queries, this demonstrates that selecting features according to query dependent weights under time constraints (i.e., largest weights first), as employed by the Indep model, is more beneficial for verbose queries than short title queries. This is in line with previous findings [17, 4] for creating effectiveness-centric ranking models (i.e., no time constraint), where query dependent weighting was shown to be more useful for long queries. The Joint model consistently outperforms both QL and Indep by large margins for the description queries, and in most cases the differences are significant, which confirms our earlier observation that by reducing feature redundancy and selecting high quality features, the joint model can achieve high effectiveness across a wide range of time constraints. Note that for description queries, the Joint model outperforms the Indep model even on looser time constraints because there exists more redundancy in longer queries.

In addition to effectiveness, we are also interested in measuring how quickly each model converges to the effectiveness of the “All features” upper-bound. To approximate this measure, we let  $T_d$  denote the time (relative to  $T_{QL}$ ) taken by each model to achieve  $d\%$  of the upper-bound effectiveness, where  $d\%$  is chosen to be 98% in our experiments and effectiveness is measured in MAP. In terms of  $T_{98}$ , we see that for both title and description queries, the Joint model has quantitatively better convergence rates than the Indep

model across all collections. Furthermore, unlike the Indep model, the Joint model is able to attain 98% of upper-bound effectiveness across all collections and query types. We note that the QL baseline is not able to attain the specified effectiveness (i.e., 98% of upper-bound), thus it is not meaningful to compute  $T_{98}$  for it.

#### 4.4 Comparison with SD Model

How does our temporally constrained ranking models compare with a previously proposed effectiveness-centric model? The sequential dependence model (SD) [21] is a widely-used term proximity retrieval model: it adheres to a rigid efficiency-effectiveness tradeoff (i.e., it does not adapt to time constraints) but has demonstrated good performance across various tasks [21, 17, 4]. For fairness of comparison, we calibrate the time constraints of our temporally constrained models with respect to the SD model’s time  $T_{SD}$ , where  $T_{SD}$  is approximately equal to  $4 \cdot T_{QL}(q)$  for each query  $q$ . Table 6 reports results on title and description queries for the SD, Indep, and Joint models.

From these results, it is evident that both the Indep and Joint models significantly outperform SD under the same temporal constraint as the SD model. For title queries, the gains in MAP range between 4.1% and 5.4% for Indep, and range between 5.0% and 10.7% for the Joint model. Further, we observe that the gains of Indep over SD are greater for description queries than title queries. For example, the Indep model improves over SD by 11.27% and 11.7% for Wt10g and Gov2, respectively, more so than for title queries, with the only exception being the Clue collection description queries. However, the Joint model consistently outperforms both the SD and Indep models in all conditions. These results not only suggest that our temporally constrained ranking functions can subsume the SD model and the QL baseline (as illustrated previously) as special tradeoff cases between effectiveness and efficiency, but our models can in fact return better results than those commonly-used retrieval models under the same time cost.

	Title Queries			Description Queries		
	Wt10g	Gov2	Clue	Wt10g	Gov2	Clue
SD	22.44	33.57	21.68	19.78	29.84	18.36
Indep	<b>23.66*</b>	35.12*	22.57	21.55	32.39*	17.67
Joint	23.66	<b>35.25*</b>	<b>24.00</b>	<b>22.01†</b>	<b>33.36*</b>	<b>18.89</b>

**Table 6:** MAP scores of SD, Indep and Joint at  $T_{SD}$  for title queries (left) and description queries (right) for Wt10g, Gov2, and Clue. The \* and † represent sig. differences with respect to SD and Indep, respectively.

## 5. FUTURE WORK AND CONCLUSIONS

In this paper, we introduced the notion of temporally constrained ranking functions for information retrieval, which, given a query and a time constraint, produces the best possible ranked list within the specified time limit. This new problem was motivated by the fact that although current learning-to-rank approaches for information retrieval are capable of learning highly effective ranking functions, the important issue of efficiency has been largely ignored. Our models can be viewed as a way to control the effectiveness/efficiency tradeoff when learning to rank. We have shown that they subsume commonly-used ranking models as special tradeoff cases that encode fixed points in the effectiveness/efficiency solution space.

We proposed and empirically evaluated two temporally constrained ranking algorithms: one that makes *independent* feature selection decisions, and the other that makes *joint* feature selection decisions. Experiments on a number of collections show that while both algorithms are effective across different time constraints, the Joint model delivers higher quality results. We also verified that our models can guarantee actual query evaluation times in practice. Furthermore, we demonstrated that our models can in fact return more effective results than those commonly-used retrieval models under the same time cost.

There are several specific future directions. First, we would like to develop a better understanding of feature redundancy and its impact on ranking functions beyond the current redundancy features examined in this work. Second, due to limited space, we were only able to show a few possibilities for time constraints. We would like to explore various distribution models on time constraints (Gaussian, exponential, etc.) to account for various real-world situations. Finally, we are interested in building and plugging in additional analytical models of query evaluation time into our general framework that can capture other retrieval strategies (e.g., those that incorporate caching).

## 6. ACKNOWLEDGMENTS

This work was supported in part by the NSF under awards IIS-0836560 and IIS-0916043; by DARPA contract HR0011-06-02-001 (GALE). Any opinions, findings, conclusions, or recommendations expressed are the authors' and do not necessarily reflect the sponsors'. The third author is grateful to Esther, Kiri, Joshua, and Jacob for their loving support.

## 7. REFERENCES

- [1] J. Allan, J. Aslam, B. Carterette, V. Pavlu, and E. Kanoulas. Million query track 2008 overview. *TREC*, 2008.
- [2] V. Anh, O. de Kretser, and A. Moffat. Vector-space ranking with effective early termination. *SIGIR*, p. 35–42, 2001.
- [3] R. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri. The impact of caching on search engines. *SIGIR*, p. 183–190, 2007.
- [4] M. Bendersky, D. Metzler, and W. Croft. Learning concept importance using a weighted dependence model. *WSDM*, 2010.
- [5] C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.
- [6] S. Büttcher, C. Clarke, and B. Lushman. Term proximity scoring for ad-hoc retrieval on very large text collections. *SIGIR*, p. 621–622, 2006.
- [7] B. Cambazoglu, H. Zaragoza, O. Chapelle, J. Chen, C. Liao, Z. Zheng, and J. Degenhardt. Early exit optimizations for additive machine learned ranking systems. *WSDM*, 2010.
- [8] D. Carmel, D. Cohen, R. Fagin, E. Farchi, M. Herscovici, Y. Maarek, and A. Soffer. Static indexing pruning for information retrieval systems. *SIGIR*, p. 43–50, 2001.
- [9] C. Clarke, F. Scholar, and I. Soboroff. Overview of the TREC 2005 terabyte track. *TREC*, 2005.
- [10] K. Collins-Thompson. Reducing the risk of query expansion via robust constrained optimization. *CIKM*, p. 837–846, 2009.
- [11] T. Dean and M. Boddy. Time-dependent planning. *AAAI*, p. 49–54, 1988.
- [12] D. Edwards. *Introduction to Graphical Modelling*. Springer, 2000.
- [13] J. Gao, J. Nie, G. Wu, and G. Cao. Dependence language model for information retrieval. *SIGIR*, p. 170–177, 2004.
- [14] X. Geng, T. Liu, T. Qin, and H. Li. Feature selection for ranking. *SIGIR*, p. 407–414, 2007.
- [15] M. Jordan. Graphical models. *Statistical Science*, 19(1):140–155, 2004.
- [16] M. Jordan, Z. Ghahramani, T. Jaakola, and L. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233, 1999.
- [17] M. Lease. An improved Markov Random Field model for supporting verbose queries. *SIGIR*, p. 476–483, 2009.
- [18] J. Lin, D. Metzler, T. Elsayed, and L. Wang. Of Ivory and Smurfs: Loxodontan mapreduce experiments for web search. *TREC*, 2009.
- [19] T.-Y. Liu. Learning to rank for information retrieval. *FNTIR*, 3(3), 2009.
- [20] D. Metzler. Automatic feature selection in the Markov Random Field model for information retrieval. *CIKM*, p. 253–262, 2007.
- [21] D. Metzler and W. Croft. A Markov Random Field model for term dependencies. *SIGIR*, p. 472–479, 2005.
- [22] J. Nocedal and S. Wright. *Numerical optimization*. Springer, 2006.
- [23] A. Ntoulas and J. Cho. Pruning policies for two-tiered inverted index with correctness guarantee. *SIGIR*, p. 191–198, 2007.
- [24] J. Ponte and W. Croft. A language modeling approach to information retrieval. *SIGIR*, p. 275–281, 1998.
- [25] S. Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. *TREC*, p. 109–126, 1994.
- [26] D. Roth and W. Yih. Integer linear programming inference for conditional random fields. *ICML*, p. 736–743, 2005.
- [27] M. Shmueli-Scheuer, C. Li, Y. Mass, H. Roitman, R. Schenkel, and G. Weikum. Best-effort top-k query processing under budgetary constraints. *ICDE*, p. 928–939, 2009.
- [28] T. Strohmaier, H. Turtle, and W. Croft. Optimization strategies for complex queries. *SIGIR*, p. 219–225, 2005.
- [29] L. Wang, J. Lin, and D. Metzler. Learning to efficiently rank. *SIGIR*, p. 138–145, 2010.
- [30] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. *SIGIR*, p. 334–342, 2001.