Ranking under tight budgets

Christian Pölitz
Saarland University
Saarbrücken, Germany
Email: poelitz@mmci.uni-saarland.de

Ralf Schenkel
Saarland University
Saarbrücken, Germany
Email: schenkel@mmci.uni-saarland.de

***Abstract*—This paper introduces a budget-aware learning to rank approach that limits the cost for evaluating a ranking model, with a focus on very tight budgets that do not allow to fully evaluate at least for one time all documents for each term. In contrast to existing work on budget-aware learning to rank, our model allows to only partially evaluate parts of the ranking model for the most promising documents. In contrast to existing work on top-k retrieval, we generate an execution plan before the actual query processing starts, eliminating the need for expensive in-memory accumulator management. We consider a unified cost model that integrates loading and processing cost. An extensive evaluation with a standard benchmark collection shows that our method outperforms other budget-aware methods under tight budgets in terms of result quality.**

***Keywords*-Learning to Rank; Constraints;**

I. Introduction

Search engines have become an important tool for the retrieval of information. A user with a complex information need expresses it as keyword query, and the search engine uses a possibly complex ranking model to retrieve the best documents that are most likely to satisfy the user. This is usually implemented by precomputing per-term information and storing it in inverted lists (or posting lists), containing postings that denote in which documents the term occurs, and at which positions. At query processing time, information from these lists is first loaded to memory (if it does not already reside there), and then fed into a ranking function that computes a score for each document (later also noted as ranker), which expresses an estimation for the relevance of that document. A number of different ranking functions have been proposed, from simple tf-idf models to recent learning-to-rank approaches.

An important observation is that most of the state-of-the-art learning-to-rank methods use computationally expensive models and ensembles of single models. For instance the most successful contributions of the 2010 Yahoo! Learning to Rank Challenge use ensembles of tree based models (see Chapelle et al. [5]). These models compute scores for documents given a query to estimate the relevance of a document by many computationally expensive models leading to an even more expensive calculation of the whole ensemble.

Since computing scores for all documents with complex models is usually too expensive even when indexes are in memory (and even more when they need to be loaded from disk), a number of evaluation algorithms exist that aim at limiting the execution effort to find the best documents. Such dynamic pruning or top-k algorithms [14], [7] consider a fixed scoring function and only a subset of all postings. In general, two classes of such algorithms exist: while the first class consumes postings in expected order of importance (e.g., using precomputed scores [6] or impacts [1], [10]), the second class dynamically skips postings based on score estimates [3]. Efficient evaluation of learning-to-rank models has not been equally well studied so far. Current state-of-the-art approaches are described in [4] and [12]. Further, precomputing all score values would be expensive that standard top-k methods cannot be applied.

The problem gets an additional twist when the cost for processing a query (e.g., measured in terms of time or disk accesses) is limited by a fixed *budget*, requiring to find the best possible (not necessarily the best overall) results within the given budget. The vast majority of retrieval algorithms are budget-oblivious, optimizing only for finding the best overall results. Only a few *budget-aware* algorithms have been proposed. Shmueli-Scheuer et al. [9] extended a standard top-k algorithm with scheduling heuristics to optimize quality of intermediate results. Wang et al. [13] introduced budget-awareness for learning-to-rank methods, optimizing parameters for varying budgets. To keep budgets at runtime, their approach evaluates only a subset of all features on all documents. Pölitz et al. [8] extended the approach by Wang et al. to evaluate not only subsets of features but also subsets of documents.

We propose a joint optimization framework that, given a cost budget and precomputed posting lists, decides which postings to read and how to evaluate them. Compared to budget-aware, top-k algorithms, we can exclude processing that do not contribute much to result quality, saving executing cost. Compared to budget-aware learning to rank, we can exclude postings that are unlikely to belong to the final result, again saving execution cost. This is especially useful under tight budget constraints that do not allow to fully evaluate all documents for all terms, an important case not considered by Wang et al. [13].

First we introduce a cost model with two components, *loading* and *computational* costs. While the former measures

the effort for loading postings, the latter measures the effort to compute the ranking. Our optimization framework uses a joint model of loading and processing costs and estimates the usefulness of loading postings and processing them. We define logical dependencies for processing different parts of the ranking method. By this we can reasonably calculate the costs of using certain parts exclusively in order to keep given budgets. We parameterize a ranking model with the most important postings and score calculations that can be used while keeping the budgets.

II. Related work

The problem of assigning a fixed budget for the execution of a ranking while expecting good retrieval quality has recently emerged in information retrieval. We distinguish two main approaches. First, heuristics based models that pose assumptions about the distributions of relevant information and their influence on the ranking process. Second, learning-to-rank based models that try to learn the optimal model over training data.

As a heuristic approach Shmueli-Scheuer et al. [9] describe a budget aware top-k algorithm. They define an accesses plan on sorted index lists, hence lists of evaluated posting information, used for a top-k query processing. This plan is limited on its number of accesses by a given budget. To find the best plan they assume that large list values at the beginning increase the possibility to find the top-k results early. Later on, strongly decreasing values help to determine the final top-k result set faster. The idea to find an optimal access plan to minimize the accesses on index lists was already introduced by Bast et al. [2]. The authors estimate the score distribution over precomputed index lists. Based on this they distribute the accesses among the lists. This approach does not provide the best results for a given budget of accesses, it minimizes only the number of accesses.

Cambazoglu et al. [4] propose to selectively put pruning/filter functions (early exit functions) between ranker (boosted decision trees) in an additive ensemble (or chain) of rankers. In the chain of the execution of rankers they place filter functions to reduce the number of document candidates on the fly. After each early exit function the number of document candidates is reduced and the following rankers need less execution time. The approach can be extended to be budget aware. The early exit functions can be parameterized and placed in such a way that their execution time is restricted to a given budget.

Wang et al. use learning-to-rank methods to leverage ranking methods in favor either for ranking quality or execution time. While in [11] they explored the trade-off between efficiency and effectiveness, hence quality and run time, in [13] they learn a model that retrieves optimal ranking results under a given budget. Only the most important features for the ranking are used in order to keep a given budget. The importances are learned from a training set. The disadvantage of their proposed method is that they exclude whole features and whole terms in order to keep a budget. By this they possibly toss important information under very tight budgets. Wang et al. [12] introduce a cascade of pairs of pruning functions and weak rankers. The cascade applies successively document pruning and features (BM25 and Dirichlet) to produce a final set of top k documents for a given query. The weak rankers are placed in increasing order of their complexity that the most complex ones need only to process a small set of documents due to the many previous prunings. The construction of the cascade is learned form training data by optimizing a trade off measure of efficiency and effectiveness.

[4] and [12] do not comply to very tight budgets if we cannot even load all necessary data at the beginning.

The differences of our work to these related works are the following, (1) we can explicitly distinguish between loading and processing costs. Most notable (2) we are able to perform optimal rankings w.r.t. quality under very tight budgets. Finally (3) our model complies to modern efficiently implemented ranking methods.

III. Scoring model

We use an additive ensemble of base rankers (see Equation 1) to find the best ranking of documents for a given query. These base rankers $F_i(q,.)$ express how good the query matches a document d. Examples of base rankers are gradient boosted decision trees as in [4] or single features as in [12]. Every base ranker estimates the relevance of the document to the query. They are applied to subsets $S(q)$ of terms from the query q. The concrete used subsets for a given query q are terms and consecutive term pairs. We note $q' \in S(q)$ with $q' = t_i$ for terms and $q' = t_i t_{i+1}$ for bigrams.

$$score(q,d) \quad = \quad \sum F_j(S(q),d) \tag{1}$$

IV. Parameterization

The goal of our method is to perform a ranking with the above introduced model that is able to consider a budget constraint. Adhoc, we decide which parts of the model shall be used for the current ranking task. There are three main components of our model. First the base rankers, second the subsets of the query that are used and the documents. For these components we introduce a set of binary variables X that are used to restrict the model to those parts of the components having the corresponding variable set to one. For the base rankers there are the variables X_j. Further each subset $q' \subseteq q$ is associated with the variable $X_{q'}(d)$. These variables additionally depend on the documents d. This means setting $X_j = 1$ initializes the model to (additionally) use base ranker j or $X_{q'}(d) = 1$ to evaluate document d for the subset q'. Equation 2 shows the parameterized version of our above introduce ranking model. Depending in the variables, the model uses base rankers on subsets of the

query to estimated the relevance of the document for the query.

$$\begin{aligned} score(q,d,X) &= \sum F_j(S(q,d,X),d) \cdot X_j \quad (2) \\ S(q,d,X) &= \{q' | q' \subseteq 2^q \wedge X_{q'}(d) = 1\} \end{aligned}$$

V. Data Structures

From the parameterized ranking model as defined in Equation 2 we need to process all subsets $S(q,d)$ and estimate the relevance for documents d. For the terms t_i this is done by processing positional information which are stored in posting lists L_{t_i}. The lists L_{t_i} contain, for each document containing t_i, its document id and the list of positions where the term appears. We do not precompute posting lists for bigrams $t_i t_{i+1}$, but process all the postings of the contained terms t_i, t_{i+1} and use the positional information to detect bigram cooccurrences.

We separate the postings of a term into classes w.r.t. loading effort and importance of documents. The importance of a document is estimated by its impact, similar to [1]. We separate impact ordered posting lists into disjoint partial lists $L_{t_i,k}$, resp. blocks. Within these blocks the postings are sorted by document id, resulting in better compression. This results in separately loadable partial posting lists. The logical variables are extended to $X_{t_{i,k}}$ expressing that for term t_i only postings from class k shall be used. For bigrams $t_i t_{i+1}$, posting class $L_{t_i t_{i+1},k,k'}$ corresponds to all bigram postings that can be constructed from positional information in posting classes $L_{t_i,k}$ and $L_{t_{i+1},k'}$.

VI. Documents and Postings Selection

To reduce processing and loading effort we want to use only a subset of documents from the posting lists for the relevance estimation. Using the introduced posting classes we define the usefulness of the documents in these classes and the costs to expect when loading them.

A. Usefulness

We define the usefulness of a term t_i of the query as inverse document frequency $IDF(t_i)$. We denote the importance of posting class k as $\delta_k \cdot IDF(t_i)$. The parameter δ represents the weight of a posting class. Additionally, since a bigram uses two terms t_i and t_{i+1} restricted to posting class k for t_i and k' for t_{i+1}, the importance of the corresponding posting classes for the bigram is $\delta_{kk'} \cdot IDF(t_i t_{i+1})$. This introduces additional parameters δ which have to be set optimally. Later we will explain how to find optimal values for all parameters. We define the usefulness $U(q')$ of subquery q' when using certain posting classes for terms and bigrams in Equation 3.

$$\begin{aligned} U(t_i,X) &= \sum_k \delta_k \cdot IDF(t_i) \cdot X_{t_{i,k}} \quad (3) \\ &+ U(t_i t_{i+1}) + U(t_{i-1} t_i) \\ U(t_i t_{i+1},X) &= \sum_{k,k'} \delta_{kk'} \cdot IDF(t_i t_{i+1}) \cdot X_{t_{i,k}} \cdot X_{t_{i+1,k'}} \end{aligned}$$

From the definition of the usefulness we can see that using a bigram will start giving use when we will also use posting classes of the contained terms. Further using a term is more useful when additionally using terms to form a bigram. In this case we add the usefulness of the corresponding bigrams to the usefulness of the term.

B. Costs

The cost of using a postings class is set to the number of postings to be loaded, hence those being in the corresponding classes to be used, multiplied by a constant factor k_l. The factor k_l is used as measure for the effort of loading one posting. Generally we distribute the costs in a top down manner as described in Equation 4. For bigrams no loading costs occur since they use already loaded term posting lists.

$$\begin{aligned} costs_l(t_i t_{i+1},X) &= 0 \quad (4) \\ costs_l(t_i,X) &= \sum_k costs_l(t_i,k) \cdot X_{t_{i,k}} \\ costs_l(t_{i,k}) &= |L_{t_{i,k}}| \cdot k_l \end{aligned}$$

VII. Base Ranker Selection

The base rankers for our ranking model as described above depend on the subsets they are applied to and also on each other. For each loaded document and each term, the final score is calculated in one shot. This means we perform a document-at-a-time processing during a join on the posting lists. For bigrams, positional information for the contained terms must be traversed once to compute cooccurances and the score.

A. Usefulness

The usefulness of a ranker consists of two parts. The weights ϵ and a decay parameter ρ of the base rankers indicating their usefulness. The more ranker we apply to a document the less influence on the usefulness is to expect. Hence, the quality of the ranking results will only slightly increase when using additional base rankers. The decay parameter reduces the usefulness of the base ranker depending how many other base ranker $(t-1)$ are placed in the ensemble before it (see Equation 5).

$$U(F_j) = \epsilon_j \cdot \rho_t \quad (5)$$

B. Costs

The processing of the documents by the rankers imposes computational costs. There are two aspects to be considered. First, for each term, postings must be traversed to process it. Second, for each base ranker to be applied, some additional processing cost is needed. Hence at each step in the traversal of the postings a constant k_p for processing the current posting and constant k_r for applying the ranker on the current posting is added to the costs. For bigrams, we only need to consider traversing the postings of two terms. This results in nonlinear costs from the first to the second

application of a ranker on the same document, from the second on the costs are linear. The definition is given in Equation 6.

$$costs_p(F_j, X) = (k_p \cdot I(F_{j' \neq j}) + k_r) \cdot \sum_{X_{t_i,t_{i+1},k}=1} (|L_{t_{i,k}}| + |L_{t_{i+1,k}}|) \cdot X_j + (k_p \cdot I(F_{j' \neq j}) + k_r) \cdot \sum_{X_{t_i,k}=1} |L_{t_{i,k}}| \cdot X_j \quad (6)$$

The function I returns one if any other ranker is already set to be used. In this case only the costs for applying the ranker j occur, no costs for the processing of the postings is further charged.

VIII. Unified Model

Assuming a budget B restricting the run time. We need to optimally distribute the budget on loading and processing. Consequently the usefulness and the costs of using posting classes and rankers must be combined, see Equation 7.

$$cost(L_{q'}, F_j, X) = costs_l(L_{q'}, X) + costs_p(F_j, X) \quad (7)$$

The usefulness of a posting class of a term and the application of a ranker can be simply expressed as product of their individual usefulness as defined in Equation 8.

$$U(q', F_j) = U(q', X) \cdot U(F_j) \quad (8)$$

To build a unified model as claimed in the beginning we need to be able to restrict both the loading and the computational costs w.r.t. to a given budget B. This means we need a combined optimization to find the optimal posting classes to be loaded and the optimal rankers to be applied w.r.t. the budget. We can formulate an optimization task using loading and computational costs while utilizing the dependencies among the subsets and the rankers, see Equation 9. To force that bigrams can only be used when the corresponding terms are used as well, we introduce the additional constraint $X_{t_i t_{i+1}} = X_{t_i} \cdot X_{t_{i+1}}$.

$$X = argmax_{X'} \sum_{q' \in S(q), j} U(q', F_j) \cdot X_j \cdot X_{q'} \quad (9)$$

$$s.t. \quad \sum_{q' \in S(q)} costs_l(q', X') + \sum_j costs_c(F_j, X') \leq B$$

To solve this optimization problem we use a greedy algorithm that continuously adds currently most useful base rankers or posting classes from a candidate set.

A. Optimal model parameters

As stated above our ranking model depends on many parameters. The weights of the posting classes, the weights of the base rankers and the decay factor. In order to set them optimally we learn them by solving an optimization problem. We use a given training data set containing documents D, a set of queries Q_{tr} and relevance information telling which documents are relevant for the individual queries and which not. Based on the relevance information an evaluation metric assesses the ranking. We look for those parameter values that maximize this evaluation metric E over a training set of queries Q_{tr} for budgets B. We follow the approach by Wang et al. [13] and use a line search optimization to solve Equation 10.

$$argmax_{\epsilon,\omega,\delta,\rho} \frac{1}{|Q_{tr}|} \cdot \sum_{q \in Q_{tr}} \sum_B E(D, score_{X(B)}(q, .)) \quad (10)$$

IX. Experiments

We implemented and applied our approach to the TREC .Gov2 data set from the TREC Terabyte track, with topics 701-850, as well as TREC WT10g data set, with topics 451-550 (using titles only). The parameters ω, ϵ, δ and ρ of the ranking model are learned by maximizing the Mean Average Precision on topics 701 to 775 for .Gov2 and topics 451 to 500 for WT10g. All tests are done on the remaining topics 776 to 850, resp. 551 to 600.

For training and testing we used budgets on the time in ms. We restricted all our tests to budgets up to 1000 ms for the .Gov2 data and up to 80 ms for the WT10g data. Further we used 10 posting classes. All significant test were performed by the Wilcoxon Signed Rank Test with $p < 0.05$.

We choose two previous related approaches for the validation of our methods: Cambazoglu et al.'s Early Exit Optimization [4] and Wang et al.'s Cascade Model [12]. Since both approaches do not explain how to efficiently find enough documents candidates in the very beginning, we combine their approaches with a top k with $k = 1000$ method for the first stage of the ensemble, resp. the cascade. Additionally we used a simple top k [7] with $k = 20$ method with BM25 features as baseline. We use the same rankers as in [12] - BM25 and Language Model with Dirichlet prior based features. Other rankers are also possible but is not the focus of this work. For the concrete experiments we used the configuration provided by Wang et al.[1] and Early Exit functions with rank thresholds like Cambazoglu et al. We tuned all methods in such a way that we get the best possible results while keeping a given budget for at least 95% of the test queries.

All methods are implemented in a comparable manner. Documents are loaded and (for our method) all rankers w.r.t. the parameterization are applied in one run, or (for Cambazoglu et al. and Wang et al.) only the first ranker is applied and the best 1000 documents are maintained and later processed by the rest of the ensemble resp. cascade. For the top 20 method additional candidate pruning is performed.

The charts in Figure 1 and Figure 2 show the NDCG achieved by the tested methods under different budgets. The top 20 method needs in our implementation 70 ms for WT10g and 900 ms for .Gov2. Hence, for smaller budgets

[1] http://www.umiacs.umd.edu/~jimmylin/ivory/docs/index.html

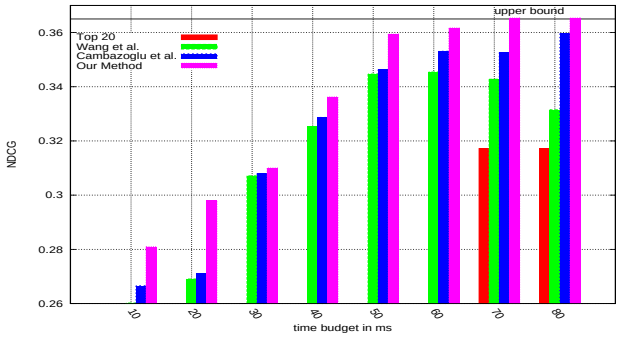

Figure 1. NDCG for different budget on the WT10g data set.

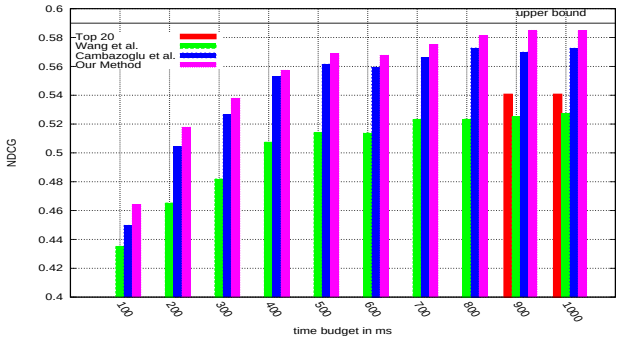

Figure 2. NDCG for different budget on the .Gov2 data set

this method cannot be used. Further, we show the upper bound of the NDCG at 20. This is the value we can achieve if we set the budget to infinity. For both data sets our proposed method shows better results compared to the other methods. Cambazoglu et al.'s Early Exit Optimization performs not as good as our method, but outperforms Wang et al.' Cascade model. The Cascade model shows relatively poor results - for .Gov2 even worse than Top 20.

In Table I we summarize the results of our experiments. We report average NDCG at 20 and Precision at 20 over all tested budgets as well as how many times we could actually keep the budget. Our method outperforms all other methods in terms of retrieval quality. On the WT10g data set we get 5% better results w.r.t. NDCG. For Precision at 20 we get only slightly better results. On the .Gov2 data set we get 10% better results compared to Wang et al's method. Compared to Cambazoglu et al.'s we get 2 to 3% better results.

Table I
MEAN NDCG@20 AND PRECISION@20 OVER ALL TESTED BUDGETS. ERROR NOTES HOW MANY TEST QUERIES COULD NOT ACTUALLY END BEFORE THE BUDGET WAS EXCEEDED. BOLD NUMBERS SHOW BEST RESULTS FOR THE DATA SETS. *SHOWS SIGNIFICANT IMPROVEMENTS.

	WT10g			.Gov2		
Method	Error	NDCG	P20	Error	NDCG	P20
Cascade	2%	31.58	25.96	3%	50.17	46.51
Early exit	2%	31.57*	26.13*	4%	54.35*	50.75*
Our method	3%	**33.40***	**26.64***	4%	**55.39***	**52.07***

X. CONCLUSIONS AND FUTURE WORK

We explained the problem of ranking under budgets on loading and computational costs. We introduced dependencies of documents and rankers. The dependencies were used to estimate the use we gain when using postings of certain terms and applying specific rankers. Further the dependencies were applied to calculate costs that occur when loading and processing postings. We defined an optimization task of which the solution results in the optimal documents (resp. postings) to load and rankers to apply when facing an arbitrary budget jointly on loading and computational costs. The evaluation of our proposed solution of the optimization task showed better results compared with state-of-the-art budget aware ranking methods. Especially under very tight budgets we showed that our proposed method produces very good results.

For the future we plan to test our method on further data sets and in different settings. We want to investigate the influence of the number of posting classes on the quality under the budgets. Further we want to explore how much the usefulness of bigrams depends on how many postings of the terms from the bigram can be loaded.

REFERENCES

[1] Vo Ngoc Anh and Alistair Moffat. Pruned query evaluation using pre-computed impacts. SIGIR '06, pages 372–379.
[2] Holger Bast, Debapriyo Majumdar, Ralf Schenkel, Martin Theobald, and Gerhard Weikum. Io-top-k: Index-access optimized top-k query processing. VLDB '06, pages 475–486.
[3] Andrei Z. Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Y. Zien. Efficient query evaluation using a two-level retrieval process. CIKM '03, pages 426–434.
[4] B. Barla Cambazoglu, Hugo Zaragoza, Olivier Chapelle, Jiang Chen, Ciya Liao, Zhaohui Zheng, and Jon Degenhardt. Early exit optimizations for additive machine learned ranking systems. WSDM '10, pages 411–420.
[5] Olivier Chapelle and Yi Chang. Yahoo! learning to rank challenge overview. *Journal of Machine Learning Research - Proceedings Track*, 14:1–24, 2011.
[6] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.
[7] Ihab F. Ilyas et al. A survey of top- query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4), 2008.
[8] Christian Pölitz and Ralf Schenkel. Learning to rank under tight budget constraints. SIGIR '11, pages 1173–1174.
[9] Michal Shmueli-Scheuer, Chen Li, Yosi Mass, Haggai Roitman, Ralf Schenkel, and Gerhard Weikum. Best-effort top-k query processing under budgetary constraints. ICDE '09, pages 928–939.
[10] Trevor Strohman and W. Bruce Croft. Efficient document retrieval in main memory. SIGIR '07, pages 175–182.
[11] Lidan Wang, Jimmy Lin, and Donald Metzler. Learning to efficiently rank. SIGIR '10, pages 138–145.
[12] Lidan Wang, Jimmy Lin, and Donald Metzler. A cascade ranking model for efficient ranked retrieval. SIGIR '11, pages 105–114.
[13] Lidan Wang, Donald Metzler, and Jimmy Lin. Ranking under temporal constraints. CIKM '10, pages 79–88.
[14] Justin Zobel and Alistair Moffat. Inverted files for text search engines. *ACM Comput. Surv.*, 38(2), 2006.