Relevance-aware Filtering of Tuples Sorted by an Attribute Value via Direct Optimization of Search Quality Metrics

Nikita Spirin[1], Mikhail Kuznetsov[2], Julia Kiseleva[3], Yaroslav Spirin[4], Pavel Izhutov[5]
[1]UIUC, Urbana IL, USA; [2]MIPT, Dolgoprudny, Russia; [3]Eindhoven University of Technology, Eindhoven, Netherlands; [4]Datastars, Moscow, Russia; [5]Stanford University, Palo Alto CA, USA
spirin2@illinois.edu[1], mikhail.kuznecov@phystech.edu[2], j.kiseleva@tue.nl[3], izhutov@stanford.edu[5]

ABSTRACT

Sorting tuples by an attribute value is a common search scenario and many search engines support such capabilities, e.g. price-based sorting in e-commerce, time-based sorting on a job or social media website. However, sorting purely by the attribute value might lead to poor user experience because the relevance is not taken into account. Hence, at the top of the list the users might see irrelevant results. In this paper we choose a different approach. Rather than just returning the entire list of results sorted by the attribute value, additionally we suggest doing the relevance-aware search results (post-)filtering. Following this approach, we develop a new algorithm based on the dynamic programming that directly optimizes a given search quality metric. It can be seamlessly integrated as the final step of a query processing pipeline and provides a theoretical guarantee on optimality. We conduct a comprehensive evaluation of our algorithm on synthetic data and real learning to rank data sets. Based on the experimental results, we conclude that the proposed algorithm is superior to typically used heuristics and has a clear practical value for the search and related applications.

Keywords

Search Metric; Attribute; Filtering; Dynamic Programming

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Information filtering, Retrieval models, Search process, Selection process

1. INTRODUCTION

Many search engines support sorting of the search results by an attribute value, e.g. sort items by price in e-commerce or sort resumes by the update time on the job websites. A similar scenario exists in the social domain when the goal is to construct a chronologically sorted social feed, e.g. Twitter, Facebook. However, sorting purely by the attribute value might not be the best approach since at the top of the list users might find irrelevant results. For example, see the screenshots of the search user interfaces for Indeed.com and Amazon.com on Figure 1. In both cases the results sorted by the attribute values are hardly relevant for the queries.

To evaluate how such search scenarios are supported today, we conducted the ad hoc evaluation of ten popular search engines from the e-commerce and job industries[1]. For each search engine we submitted 25 queries (different queries for different industries), applied the sorting based on one of the attributes (relevance, date, price), and judged the quality of results[2]. The ranking by relevance is of very high quality. The average Precision@10 is 0.86. On the other hand, we found that the search results are far from relevant when the attribute-based sorting is done. For instance, across the sites the average Precision@1 is 0.44, Precision@5 is 0.45, and 61% of queries have the Precision@10 below 0.5. We think that it is mainly due to the relevance not being taken into account when the attribute-based sorting is requested. Therefore, our research questions are: (RQ1) Can the quality of results sorted by the attribute value be improved by incorporating the relevance into the ranking process? (RQ2) What is the best way to accomplish it?

In this paper we propose a new principled approach to perform relevance-aware search results (post-)filtering via *direct optimization of a given search quality metric*. Our algorithm uses the ideas from dynamic programming and is guaranteed to deliver the optimal solution. The algorithm is presented in Section 3. The experiments on synthetic and real learning to rank (L2R) data sets are described in Section 4.

2. RELATED WORK

This work is related to the research on search user behavior analysis, search metrics, and learning to rank. The proposed algorithm is based on the dynamic programming [1].

Researchers studied the way people interact with search engines by analyzing mouse movements, eye-tracking and click logs. Joachims et al. [9] discovered the *position bias* phenomenon, i.e. the results at the first two positions receive most attention, and then it quickly drops. Plus, on average users tend to read the results in a linear order from top to bottom. Craswell et al. [4] explored how the position bias might arise and proposed four hypotheses and the corresponding probabilistic click models. They found that the "cascade" model, where users view results from top to bottom and leave as soon as they see a worthwhile document, is

[1]Amazon, Walmart, Target, Etsy, BestBuy, NewEgg for products and Indeed, LinkedIn, SuperJob, Monster for jobs.
[2]we don't describe the exact setup due to the page limit.

SIGIR'15, August 09 - 13, 2015, Santiago, Chile.

DOI: http://dx.doi.org/10.1145/2766462.2767822.

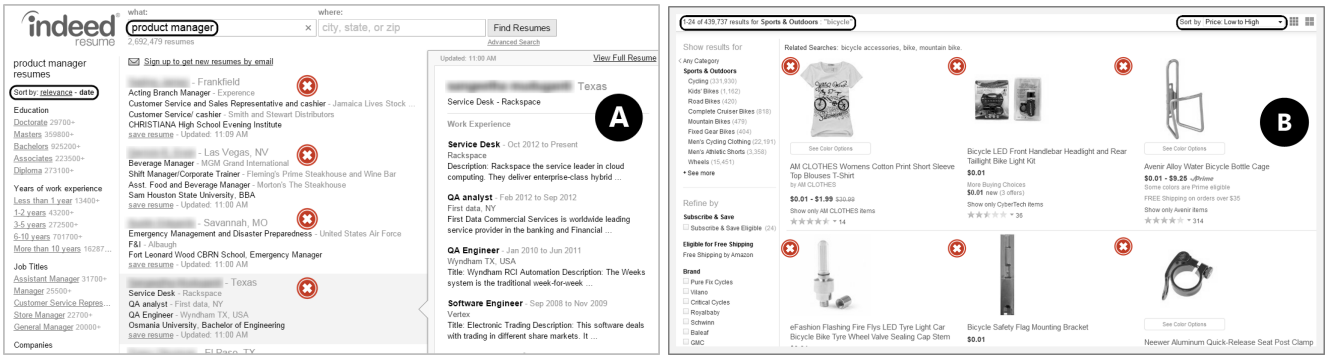

Figure 1: ***(A)*** **Indeed.com resume search results for the query** ***"product manager"*** **sorted by** ***"date"*** **and** ***(B)*** **Amazon.com search results for the query** ***"bicycle"*** **sorted by** ***"price"*****. While sorting by relevance is accurate, the results sorted by the attribute value are hardly relevant for the query, which leads to poor user experience.**

the best explanation for position bias in early ranks. Dupret et al. [5] generalized this model by allowing for the possibility that a user skips a document without examining it.

Complementary to the work on search models, a lot of attention has been devoted to the design and analysis of search metrics. Thus, in addition to the traditional metrics, like the Precision and the Recall, Järvelin and Kekäläinen proposed the (Normalized) Discounted Cumulative Gain (DCG) [8], Chapelle et al. — the Expected Reciprocal Rank (ERR) [2], to name just a few. Recently, Chuklin et al. [3] developed a unified framework to convert any click model into the evaluation metric. Essentially, all search metrics model the position bias and penalize the top ranked irrelevant results.

Numerous ranking algorithms have been developed to accurately predict the relevance of documents. Typically, these algorithms are based on machine learning and find the optimal parameters by optimizing the "surrogate" objective function. However, the solution to the approximation is not always optimal for the original ranking problem. Therefore, recently several approaches have been proposed that directly optimize a given search metric. For instance, Xu et al. [12] focus on the algorithms that optimize the objectives upper-bounding the original non-smooth search metrics. Tan et al. [11] proposed *DirectRank*, which is based on the iterative coordinate ascent with the smart line search procedure.

Attribute-based ranking, however, has been handled very differently. Rather than taking the relevance into account, search engines return the list of results sorted by the attribute value or suboptimal heuristics are used (Section 4.1). Inspired by the recent advancements in L2R, in this work we bridge the gap between the relevance-based ranking and the attribute-based ranking by proposing to do relevance-aware search results filtering, which directly optimizes a given search metric, when the sorting by the attribute value is requested. It is worth highlighting the difference between the proposed algorithm and a famous TA algorithm by Fagin et al. [6]. While TA algorithm finds the top-k most relevant tuples by scoring them *individually*, we return the tuples, which *cumulatively* optimize a given search quality metric. The ordering of the tuples is as crucial as their relevance.

3. OUR APPROACH

We consider the scenario when a user requests the sorting of the search results by the attribute value, e.g. by date (Figure 1, A) or by price (Figure 1, B). Our goal is to produce the final ranking that both satisfies the strict ordering constraints and optimizes a given search quality metric (in turn it minimizes the user's effort on finding relevant results). We only focus on the results filtering and assume that the relevance scores are already predicted by the ranking algorithm. Therefore, the formalization of our problem looks as follows.

Input: *a list of tuples* $\{(t_i, r_i)\}_{i=1}^{l}$*, where* t_i *is the attribute value and* $r_i \in \mathbb{R}+$ *is the relevance score predicted by the ranking algorithm; a search quality metric* Q*.*

Output: *a (sub)list of indices* J *delivering the maximum to the metric* Q *and totally ordered based on the attribute value, i.e.* $J = \arg\max Q(r_{j_i} | j_i \in J)$*, s.t.* $t_{j_1} < ... < t_{j_l}$*.*

Throughout the paper, we consider the DCG as the search quality metric (although ERR or other metric can be used), date as the attribute, and the input sorted chronologically. It is worth mentioning that the formalization above covers the post-filtering scenarios as well, i.e. the input might consist of tuples that passed some other filtering algorithm.

Currently, this problem is solved heuristically. Mainly there are two approaches built around the same idea of thresholding. We can take only the results that have the relevance score above the threshold. We can also sort the results by the relevance score, take the top-k elements, and finally re-sort the list by date. While these approaches are easy to implement, they have two major drawbacks. First, it is not clear how to set the threshold. Second, the described approaches are the approximate solutions of our problem. Even the result set constructed from the top-k tuples sorted by relevance, being re-sorted by the attribute value, gets ordered randomly if we look at the relevance component.

The solution that guarantees optimality is to enumerate all possible subsequences, compute the metric for each one, and take the best one. However, this approach is not tractable as the number of subsequences is exponential. We propose a polynomial algorithm based on the dynamic programming [1]. There are three key observations behind our algorithm: (1) natural enumeration order for subsequences; (2) additivity of the metric; (3) optimality of subproblems.

First, all subsequences can be partitioned into the factor classes based on their length, i.e. in each factor class there will be the subsequences of the same length. To enumerate all subsequences, we can iterate over the factor classes and

Algorithm 1 (A1) Relevance-aware filtering of totally ordered set via direct optimization of a search quality metric

Input: DCG and $\{(t_i, r_i)\}_{i=1}^{l}$, s.t. $t_i < ... < t_l$ and $r_i \in \mathbb{R}+$
Output: $J = \arg\max DCG(r_{j_i} | j_i \in J)$, s.t. $t_{j_1} < ... < t_{j_l}$

1: $M \leftarrow Matrix(l+1, l+1); M(:,0) \leftarrow 0; M(0,:) \leftarrow 0;$
2: $Path \leftarrow Matrix(l+1, l+1)$; # *to recover max sequence*
3: **for** i in $1, \ldots, l$
4: **for** j in $1, \ldots, i$
5: $gain \leftarrow \frac{2^{r_i}-1}{\log(j+1)}$;
6: **if** $M(i-1, j-1) + gain > M(i-1, j)$
7: $M(i,j) \leftarrow M(i-1, j-1) + gain$;
8: $Path(i,j) \leftarrow (i-1, j-1)$;
9: **else**
10: $M(i,j) \leftarrow M(i-1, j)$;
11: $Path(i,j) \leftarrow (i-1, j)$;
12: $(i,j) \leftarrow \arg\max M(l,:)$; # *last element of solution*
13: $J \leftarrow List(); J.append(j)$;
14: **while** $i > 1$ and $j > 1$
15: **if** $P(i,j).last < j$
16: $J.append(P(i,j).last)$;
17: $(i,j) \leftarrow P(i,j)$; # *jump to shorter subsequence*
18: **return** $J.reverse()$

within each factor class enumerate all subsequences. Second, the search metrics are additive and can be computed in linear time from the beginning of the list to the end [2]. It means that having a partial metric value for the prefix, we can compute the new metric value by simply adding the gain/utility provided by the current element. Third, the optimal subsequence for the prefix of length k is one of the optimal subsequences from each of the factor classes for the prefix of length $k-1$ with or without the current element appended (proof by induction for the prefix length).

Combining the observations above, we present our algorithm and its analysis. It starts by initializing the memoization matrix to store the optimal DCG values for subproblems and the transition matrix to reconstruct the optimal subsequence. Then, it iterates over the prefixes of the input sequence in the outer loop and over the factor classes in the inner loop. The cell (i,j) is for the optimal subsequence of length j for the prefix of length i. At each step we decide whether we should append the current element of the input sequence i to the optimal subsequence of length $j-1$ for the prefix of length $i-1$ (the recursion on lines 6-11). If we append the current element, we go diagonal. If we don't append, we keep the existing optimal subsequence of length j and stay on the same column. A legal evaluation order and the dependencies between the cells are shown in Figure 2, A. Finally, to reconstruct the optimal subsequence, we find the maximum in the last row (the last element is always "in" since the elements are non-negative) and go backwards in the $Path$ matrix. If the line is diagonal, we take the element in the next cell. Otherwise, we skip. The $Path$ matrix is depicted in Figure 2, B. The complexity (both time and space) of the algorithm is $O(l^2)$ because we have two nested loops, costing us $O(1)$ time at each iteration, and the square memoization matrices. It is guaranteed to deliver the optimum because we "virtually" enumerate all subsequences within the dynamic programming framework. For a toy example problem $\{(0,0),(1,3),(2,1),(3,2),(4,1),(5,3))\}$ the optimal solution is $\{1,3,4,5\}$ with the DCG equal to 12.40.

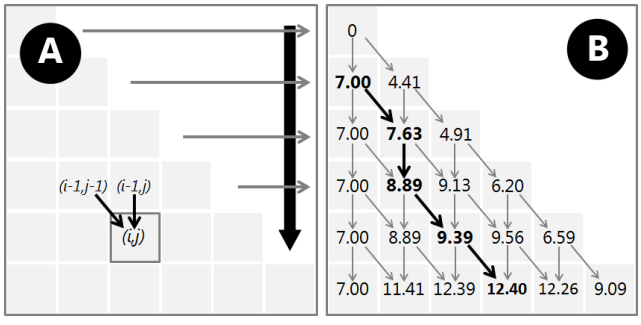

Figure 2: Dependencies in the memoization matrix, a legal evaluation order, and the optimal path.

4. EXPERIMENTS AND RESULTS

In this section we study how our approach contributes to the ranking quality using two real LETOR [10] (MQ2007, MSLR-WEB10K) and synthetically generated data sets.

4.1 Learning to Rank Data Sets

To answer our research questions, we do the simulations using the real learning to rank data sets. We extend MQ2007 and MSLR-WEB10K data sets by assigning a random timestamp to each document to model the sorting by the attribute value. Scikit-learn[3] implementation of the Gradient Boosted Regression Trees ($GBRT$) [7] is used to predict the relevance scores. The optimal parameters for the final $GBRT$ model are picked using cross validation for each data set. We use the 5-fold cross validation partitioning from LETOR [10].

Three popular baselines are considered, which are typically used to perform the filtering of the search results:
Baseline 1 (B1): sort by the attribute value, no filtering;
Baseline 2 (B2): sort by the attribute value, keep the results with the predicted relevance scores above the threshold (we normalized the scores to [0,1] and set the threshold=0.5);
Baseline 3 (B3): sort results by the predicted relevance score, take the top-k (where k is the cutoff point for the metric value calculation), and re-sort by the attribute value.

The evaluation procedure works as follows. First, we train the $GBRT$ on the training folds. Second, we predict the relevance scores using the trained $GBRT$ model for the documents in the testing fold. Third, we apply a baseline filtering algorithm to the documents in the testing fold by working with the relevance scores from the step two and the randomly generated timestamps. Fourth, we apply our filtering algorithm to the tuples that passed the baseline filtering. Finally, knowing the true relevance labels, we calculate the $NDCG@k$ for the filtered result list sorted by the timestamp. To make sure that the conclusions are not due to randomness, we average the results from 1000 runs.

The results of the experiment are presented in Table 1 and 2. We can see that the output (post-)filtered with our algorithm is regularly better than the baselines. We applied the binomial test and found that almost all differences in the $NDCG$ values are statistically significant (marked in bold), p-value is below 0.001. One average the increase in the metric value is around 2-4%. Moreover, since the data sets used have very different characteristics (e.g. the average query

[3] `http://scikit-learn.org/stable/index.html`

NDCG	*@1*	*@5*	*@10*	*@20*	*@40*
B1 only	0.226	0.245	0.273	0.336	0.496
A1 ∘ B1	**0.299**	**0.287**	**0.304**	**0.363**	**0.511**
B2 only	0.289	0.318	0.357	0.448	0.450
A1 ∘ B2	**0.315**	**0.326**	**0.364**	**0.453**	**0.454**
B3 only	0.433	0.417	0.418	0.451	0.498
A1 ∘ B3	0.433	0.417	**0.420**	**0.455**	**0.512**

Table 1: The demonstration of effectiveness of the proposed approach on MQ2007 data set.

NDCG	*@1*	*@5*	*@10*	*@20*	*@40*
B1 only	0.131	0.161	0.190	0.236	0.309
A1 ∘ B1	**0.173**	**0.183**	**0.208**	**0.250**	**0.321**
B2 only	0.170	0.208	0.244	0.300	0.379
A1 ∘ B2	**0.192**	**0.215**	**0.250**	**0.304**	**0.383**
B3 only	0.390	0.362	0.365	0.380	0.418
A1 ∘ B3	0.390	0.362	**0.366**	**0.382**	**0.421**

Table 2: The demonstration of effectiveness of the proposed approach on MSLR-WEB10K data set.

length for MQ2007 is 40 and for MSLR-WEB10K — 120), the experiment suggests that the algorithm achieves good performance for a wide range of input problems. Yet, one should note that the increase in the ranking quality comes with the extra computational cost because the complexity of our algorithm is $O(\frac{l}{\log l})$ times larger than for the baselines.

4.2 Synthetic Data Sets

In this section we focus on the filtering only (both relevance labels and timestamps are generated) and study how the algorithm behavior changes for different input sizes and relevance label distributions. We consider the following four label distributions modeling the real situations: (a) uniform integer in the range $[0, 5]$; (b) uniform real in the range $[0, 5]$; (c) power law, the slope $\alpha = 2.0$; (d) $\frac{3x^2}{125}$ with the support in the range $[0, 5]$. We generate the input lists for the filtering algorithm by sampling from the corresponding distribution. Similarly to the previous experiment, we simulate each combination of conditions 1000 times and average the runs. Only the Baseline 1 is used in this experiment for simplicity. The data from the simulation is presented in Figure 3.

There are several observations that could be made with the help of this figure. First, the output size is linearly proportional to the input size (Figure 3, C). *DCG* also grows linearly with the growth of the input size (Figure 3, A). Second, the proposed algorithm always outperforms the Baseline 1 (Figure 3, B), which is expected because we do the filtering directly optimizing a given search quality metric. Third, both the graph for the ratio of the *DCG* values and the graph for the ratio of the output sequence lengths for the proposed algorithm and the baseline monotonically converge for the longer input lists (Figure 3, B and D). This means

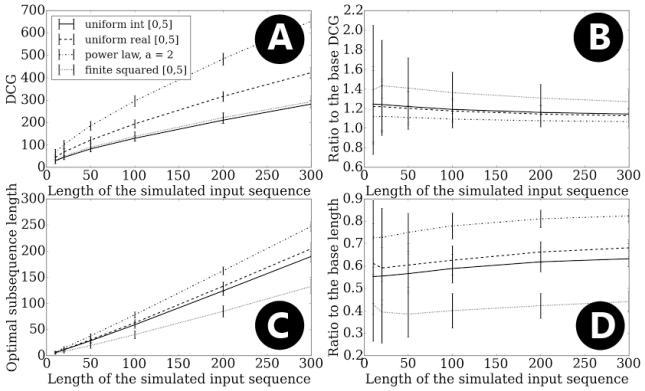

Figure 3: The behavior of the algorithm (A1) for different input sizes and relevance label distributions.

that our algorithm works better when the original hit list is shorter. Fourth, higher gains in *DCG* over the baseline are characteristic for the relevance label distributions, where relevant results are more probable (Figure 3, B). The observations above are valid for non-degenerate cases, e.g. not all labels are the same or sorted in a special order.

5. CONCLUSIONS AND FUTURE WORK

In this paper we addressed the important problem in search, that is, how to increase the relevance of the search results sorted by an attribute value. Our solution is based on the idea to perform relevance-aware search results filtering by directly optimizing a given search quality metric. We developed a simple, yet effective algorithm based on the dynamic programming, which consistently outperforms typically used heuristic approaches and is guaranteed to deliver the optimal solution. In the future, we plan to integrate the proposed algorithm in a real search engine and instrument an A/B test to see how such a modification will affect the user engagement and satisfaction with the search results.

6. ACKNOWLEDGEMENTS

We thank Karrie Karahalios, ChengXiang Zhai, and anonymous reviewers for their valuable comments and suggestions.

7. REFERENCES

[1] R. Bellman. *Dynamic Programming*. Dover Books on Computer Science, USA, 2003.
[2] O. Chapelle, D. Metlzer, Y. Zhang, and P. Grinspan. Expected reciprocal rank for graded relevance. CIKM'09.
[3] A. Chuklin, P. Serdyukov, and M. de Rijke. Click model-based information retrieval metrics. SIGIR'13.
[4] N. Craswell, O. Zoeter, M. Taylor, and B. Ramsey. An experimental comparison of click position-bias models. In *Proceedings of Web Search and Data Mining 2008*.
[5] G. E. Dupret and B. Piwowarski. A user browsing model to predict search engine click data from past observations. In *Proceedings of ACM SIGIR 2008*.
[6] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *Proceedings of PODS '01*.
[7] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5), 2001.
[8] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4).
[9] T. Joachims, L. Granka, B. Pan, H. Hembrooke, and G. Gay. Accurately interpreting clickthrough data as implicit feedback. SIGIR'05.
[10] T. Qin, T.-Y. Liu, J. Xu, and H. Li. LETOR: A benchmark collection for research on learning to rank for information retrieval. *Inf. Retr.*, 13(4).
[11] M. Tan, T. Xia, L. Guo, and S. Wang. Direct optimization of ranking measures for learning to rank models. KDD'13.
[12] J. Xu, T.-Y. Liu, M. Lu, H. Li, and W.-Y. Ma. Directly optimizing evaluation measures in L2R. SIGIR'08.