Fast Approximate Filtering of Search Results Sorted by Attribute

Franco Maria Nardini
ISTI-CNR, Pisa, Italy
francomaria.nardini@isti.cnr.it

Roberto Trani
ISTI-CNR and University of Pisa, Italy
roberto.trani@isti.cnr.it

Rossano Venturini
University of Pisa, Italy
rossano.venturini@di.unipi.it

ABSTRACT

Several Web search services enable their users with the possibility of sorting the list of results by a specific attribute, e.g., sort "by price" in e-commerce. However, sorting the results by attribute could bring marginally relevant results in the top positions thus leading to a poor user experience. This motivates the definition of the relevance-aware filtering problem. This problem asks to remove results from the attribute-sorted list to maximize its final overall relevance. Recently, an optimal solution to this problem has been proposed. However, it has strong limitations in the Web scenario due to its high computational cost. In this paper, we propose ϵ-Filtering: an efficient approximate algorithm with strong approximation guarantees on the relevance of the final list. More precisely, given an allowed approximation error ϵ, the proposed algorithm finds a (1-ϵ)–optimal filtering, i.e., the relevance of its solution is at least (1-ϵ) times the optimum. We conduct a comprehensive evaluation of ϵ-Filtering against state-of-the-art competitors on two real-world public datasets. Experiments show that ϵ-Filtering achieves the desired levels of effectiveness with a speedup of up to two orders of magnitude with respect to the optimal solution while guaranteeing very small approximation errors.

CCS CONCEPTS

• **Information systems → Retrieval tasks and goals**;

KEYWORDS

Relevance-aware Filtering; Approximation Algorithm;

ACM Reference Format:
Franco Maria Nardini, Roberto Trani, and Rossano Venturini. 2019. Fast Approximate Filtering of Search Results Sorted by Attribute. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '19), July 21–25, 2019, Paris, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3331184.3331227

1 INTRODUCTION

Many online search services provide answers to their users by estimating the relevance of the contents they deliver with respect to specific queries. The typical example of this kind of services is Web search, where users express their information needs by means of textual queries to a search engine that provides back a list of documents maximizing the relevance of the list w.r.t. the query. Other examples of services built around this approach are social networks, streaming platforms, e-commerce sites, etc. Many of these services allow the users to further interact with the provided contents. For instance, social networks allow to look for the "most recent" stories posted by friends, while shopping services provide a useful "sort by price" listing of the items matching a specific user query. These approaches work by sorting the matching results by a specific attribute, e.g., by time or by price. However, sorting merely by a specific attribute may lead to a poor user experience since the list usually contains many results that are marginally relevant for the user [14, 15]. A simple example motivates the observation above. The query "`iphone`" on *amazon.com* returns more than 100, 000 results. When results are sorted by relevance, the first page of results consists of several iPhone models and related accessories. A completely different scenario is achieved when the results are sorted by increasing price: in this case the first page of results is a list of low-price covers and gadgets. The two lists are dramatically different in terms of relevance of the results. The second list leads to a poor user experience because the user now needs to examine several items before she finds something relevant.

The problem above can be addressed as a filtering task. The goal is to retain at most k results from the original list so to maximize the overall effectiveness without changing the per-attribute sorting chosen by the user. Therefore, the Filtering@k problem is as follows.

FILTERING@k: Given a list of n relevance scores $R = \langle r_1, \ldots, r_n \rangle$, a positive integer k and a search quality metric Q, the problem asks to find the sub-list of R of size at most k that maximizes Q.

Finding a solution to the Filtering@k problem is hard when taking into account search quality metrics such as the widely-used Discounted Cumulative Gain (DCG) [6, 8, 16]. Indeed, we observe that to get an optimal solution neither it is sufficient to select the most relevant results of the list nor we can consider only the subsequences of exactly k elements. For example, given the list of four results having relevances $\langle 2, 2, 4, 1 \rangle$, the optimal DCG@3 is obtained by filtering out the first two elements, despite they are individually more relevant than the last one. Moreover, all sub-sequences of three results achieve a lower DCG@3 than the optimal one.

Recently, the Filtering@k problem has been investigated by Spirin *et al.* [14]. The authors propose an optimal solution based on a standard Dynamic Programming approach [4], which runs in $\Theta(nk)$ time. The proposed solution, hereinafter called OPT, has two important drawbacks. First, its time complexity makes its use unfeasible for online search services that needs to potentially handle several thousands of results per-query within small time budgets, e.g., 100ms for 99th-percentile per-query response time [2, 7, 9]. Second, it cannot be applied to distributed search engines as it works by taking global decisions on the complete list of results. The filtering problem is thus often addressed with two heuristics based on

SIGIR '19, July 21–25, 2019, Paris, France

ACM ISBN 978-1-4503-6172-9/19/07…$15.00
https://doi.org/10.1145/3331184.3331227

thresholding: Top$k$ and Cutoff. Top$k$ selects the k most relevant elements, while Cutoff selects the elements whose relevance is greater than a given threshold. Spirin *et al.* propose to combine Top$k$ and Cutoff with OPT to obtain two new heuristics called Top$k$-OPT and Cutoff-OPT. These two strategies are useful to trade effectiveness for efficiency. Filtering@k is thus solved by using either exact but slow algorithms or fast but inaccurate heuristics.

In this paper we contrast the weakness of the above approaches by proposing ϵ-Filtering, a novel approximate algorithm to efficiently perform the relevance-aware filtering of search results. ϵ-Filtering trades-off between efficiency and effectiveness through a parameter ϵ controlling the approximation error. It finds a (1-ϵ)–optimal filtering in $\Theta\big(n + k^2 \log_{1-\epsilon}(\epsilon/k)\big)$ time. We experimentally evaluate ϵ-Filtering on two real-world public datasets proving that it provides a significant speedup in time w.r.t. the optimal solution by Spirin *et al.* [14]. To the best of our knowledge, this is the first contribution toward defining a fast approximate algorithm for solving the relevance-aware filtering problem.

In detail, the contributions of the paper are the following:

- we start by presenting a theoretical analysis of the Cutoff-OPT and Top$k$-OPT heuristics in the filtering scenario [14]. We show that the former does not provide any performance guarantee while the latter finds a 0.5–optimal filtering in $\Theta\big(n \log k + k^2\big)$ time.
- we then propose ϵ-Filtering, a new efficient approximate algorithm with strong performance guarantees. Given an approximation error ϵ, ϵ-Filtering finds a (1-ϵ)–optimal filtering in $\Theta\big(n + k^2 \log_{1-\epsilon}(\epsilon/k)\big)$ time.
- we present a comprehensive experimental evaluation on real-world datasets. Experiments show that ϵ-Filtering is faster than Top$k$-OPT and Cutoff-OPT while providing more effective solutions. We also show that ϵ-Filtering achieves a speedup of up to two orders of magnitude w.r.t. OPT while guaranteeing very small approximation errors.

The rest of the paper is structured as follows: Section 2 discusses the related work while Section 3 presents a theoretical analysis of Cutoff-OPT and Top$k$-OPT heuristics and introduces ϵ-Filtering. Section 4 provides a comprehensive evaluation of ϵ-Filtering against OPT, Cutoff-OPT, and Top$k$-OPT on real-world data. Finally, Section 5 concludes the work and outlines future work.

2 RELATED WORK

The relevance-aware filtering problem has been addressed so far by using either optimal but slow algorithms or fast but inaccurate heuristics. We now present a review of the main contributions.

Optimal filtering. Spirin *et al.* propose an optimal algorithm, OPT, to address the relevance-aware filtering problem [14]. The authors employ dynamic programming [4] to design a $\Theta(nk)$ time algorithm to solve the filtering problem. Indeed, OPT iterates over the prefixes of the results list R to obtain optimal solutions of any length $j < k$. In this way, OPT incrementally fills a memoization matrix $M_{n\times k}$. Each entry $M[i, j]$ of the table stores the best relevance score achievable on the prefix of R of length i by using exactly j elements. At each step, OPT chooses the best decision between: i) appending the i-th element of R, r_i, to the optimal subsequence of length $j - 1$ of the prefix $i - 1$, or ii) taking the optimal subsequence of length j of the prefix $i - 1$. Specifically, it fills the table using the following recursive definition

$$M[i, j] = \max \begin{cases} M[i-1, j] \\ M[i-1, j-1] + \texttt{score\_item}(r_i, j) \end{cases}$$

where $\texttt{score\_item}(r_i, j)$ is the gain of relevance score of the element r_i when placed in position j. Clearly, the score of the empty list is 0, hence $M[\cdot, 0] = M[0, \cdot] = 0$. Once M is filled, a best score in the last row of M identifies an optimal solution. If the $\texttt{score\_item}()$ function can be computed in constant time, e.g., $\texttt{score\_item}(r_i, j) = (2^{r_i} - 1)/\log_2(j + 1)$ for the DCG metric [6], then OPT runs in $\Theta(nk)$ time.

Heuristics. Two naïve heuristics, Cutoff and Top$k$, can be employed to solve the relevance-aware filtering problem. The former selects the elements of the list whose relevance is greater than a given threshold, while the latter works by selecting the k most relevant elements of the list. Spirin *et al.* [14] experimentally showed that these two heuristics are more effective when their results are further filtered by OPT. We call Cutoff-OPT and Top$k$-OPT the resulting approaches. They also present a comprehensive evaluation on learning to rank datasets [13] and they conclude that the performance of the optimal solution is superior w.r.t Cutoff-OPT and Top$k$-OPT. However, the authors do not provide a theoretical analysis of the guarantees of the two heuristics. In this paper we provide such analysis. We first show that Cutoff-OPT does not provide any performance guarantee. We then prove that the effectiveness achieved by Top$k$-OPT may be up to two times worse than the optimum. Therefore, their application may lead to a significant degrade of the relevance of the filtered list.

Related problems. Carmel *et al.* explore a similar problem in mail search, where they propose to rank the search results of Yahoo Mail [3] by relevance. The authors investigate a mixed approach promoting the most relevant results on top of time-ranked results. Results show that supplementing time-sorted results with relevant results leads to better performance than the traditional time-sorted view. This proposal by Carmel *et al.* is only partially related to our one. While they investigate how to couple relevance ranking with time-based ranking so to enable a two-dimensional view in mail search, we propose to directly address the relevance-aware filtering of result lists by removing irrelevant results from an attribute-ordered list of results.

3 APPROXIMATE FILTERING

In this section, we first analyze the effectiveness of the two existing heuristics, Cutoff-OPT and Top$k$-OPT, proving their weak guarantees. We then introduce ϵ-Filtering, a new efficient approximate algorithm with much stronger guarantees. We present the results by using DCG [1, 6] as search quality metric Q.

3.1 Analysis of heuristics

The goal of this section is to analyze Cutoff-OPT and Top$k$-OPT. We show that Cutoff-OPT may be arbitrarily worser than OPT, while Top$k$-OPT is 0.5–optimal.

Cutoff-OPT. Cutoff-OPT applies OPT to the elements of the list whose relevance is greater than a given threshold. Cutoff-OPT does not provide any performance guarantees. Indeed, its performance strictly depends on the fixed threshold: the higher is the threshold, the more efficient and less accurate is the filtering, because an increasing number of elements is pre-filtered. It is thus trivial to find worst-case lists for Cutoff-OPT, where either i) the algorithm is not able to filter any element, or ii) it filters all of them. Therefore, in the former case Cutoff-OPT does not improve the time complexity of OPT, i.e., it runs in $\Theta(nk)$ time, while in the latter case its solution is arbitrarily worse than the optimal one.

Top$k$-OPT. Top$k$-OPT provides stronger guarantees than Cutoff-OPT. Its time complexity is $\Theta\big(n \log k + k^2\big)$ in the worst case. Indeed, it selects the top-k elements in $\Theta\big(n \log k\big)$ time. These elements are further filtered by OPT in $\Theta\big(k^2\big)$ time. As far as the quality of its solution is concerned, we prove that i) there exist an infinite family of lists for which the solution provided by Top$k$-OPT is roughly 2 times worser than the optimum, ii) this is actually the worst case, i.e., the approximation factor is always at most 0.5.

The first point is proven by using the following family of lists. We have a list R_k for each possible value of the parameter k of the Filtering@k problem. The list $R_k = \langle m, \ldots, m, 1, \tilde{m}, \ldots, \tilde{m}\rangle$ of length $2k - 1$ is made of two subsequences of equal elements of length $k - 1$ separated by 1. The value of m is $(1 - d(k))/\sum_{i=1}^{k-1} d(i)$, where $d(i)$ is the discount factor of the DCG metric, i.e., $1/\log_2(i+1)$. Instead, $\tilde{m}$ is smaller than m but infinitely close to m.

Top$k$-OPT chooses the first k elements of R_k, while OPT takes its last k elements. The scores achieved by the two strategies are 1 and $\big(2-d(k)+(1-2d(k)+d(k)^2)/\sum_{i=1}^{k-1} d(i)\big)$, respectively. For increasing values of k, the score of OPT tends to be 2 times better than the score of Top$k$-OPT, i.e., Top$k$-OPT is 0.5−optimal. For example, for $k = 20$, OPT is roughly 1.7 times better than Top$k$-OPT.

As far as the second point above is concerned, we need to prove that the worst approximation factor of Top$k$-OPT is at most 0.5 as stated by the following theorem.

Theorem 1. Top$k$-OPT is 0.5−optimal.

The proof of the theorem is omitted due to the lack of space. Here, we just give its intuition. Consider the elements that belong to the solutions computed by Top$k$-OPT and OPT. We can show that the intersection between the two solutions is non-empty, i.e., there is at least one common element. The worst effectiveness of Top$k$-OPT is achieved when the common elements are in the middle of the list R, and the elements selected by Top$k$-OPT only and OPT only are on the left and right of these common elements, respectively. Given the structure above for the list, the proof follows by appropriately choosing the relevances of its elements.

Finally, it is worth highlighting that we could not improve the above weak guarantees by increasing the number of elements selected by the Top$k$ pruning in Top$k$-OPT. Indeed, we can easily change each list R_k to match the same worst-case scenario and, thus, proving the following corollary of Theorem 1.

Corollary 2. *The solution obtained by applying OPT to the top-k' elements of R, with any $k' \geq k$, is* 0.5*−optimal.*

In conclusion, Top$k$-OPT allows to reduce the time complexity of the filtering from $\Theta(nk)$ to $\Theta(n \log k + k^2)$, but it can only guarantee a 0.5 approximation error.

3.2 ϵ–Filtering

The weak approximation guarantees provided by the heuristics motivate us to propose ϵ-Filtering, an approximate algorithm that finds a (1-ϵ)-approximation in $\Theta\big(n + k^2 \log_{(1\text{-}\epsilon)}(\epsilon/k)\big)$ time, for any $0 < \epsilon < 1$. ϵ-Filtering is composed of three steps followed by OPT: *right pruning*, *discretization*, and *thresholding*.

Right pruning. This step is a lossless pruning which filters out any element followed by at least k elements with a greater or equal relevance. The intuition behind this step is that when an element a is filtered out by the right pruning, there are at least k elements on its right that could be selected by OPT in place of a, without worsening the solution. The example below shows this intuition.

Example 3. Let R be the list $\langle 0.9, 1, \ldots, 1\rangle$ of 21 elements and $k = 20$. The element $a = 0.9$ is the only element discarded. Every solution including the element a can be improved by replacing it with a non selected element. Indeed, the solution A of k elements, $A = \langle 0.9, 1, \ldots, 1\rangle$, has DCG of about 6.9, while the solution $O = \langle 1, \ldots, 1\rangle$ has DCG of about 7.

Let right-k-maximal be any element followed by less than k elements with a greater or equal relevance. The following lemma proves the correctness of the right pruning.

Lemma 4. *There exists an optimal filtering composed of only right-k-maximal elements of R.*

Proof. By contradiction, let us assume that all existing optimal solutions contain at least one element that is not right-k-maximal. Let $\hat{O}$ be any optimal solution with the highest number of right-k-maximal elements. The contradiction arises by showing that we can replace the rightmost non right-k-maximal element of $\hat{O}$ with a right-k-maximal one without reducing the relevance of the list. Thus, showing that $\hat{O}$ is not the optimal solution with the highest number of right-k-maximal elements.

Let $\hat{r}$ be the rightmost non right-k-maximal element of $\hat{O}$ and $\hat{p}$ be its position in $\hat{O}$. By definition, there are at least k elements having relevance greater than or equal to $\hat{r}$ on its right. Furthermore, at least k of these elements are right-k-maximal. Since the size of $\hat{O}$ is at most k, at least one of them is not in $\hat{O}$. Let r be the leftmost of them, i.e., the leftmost right-k-maximal element of R located on the right of $\hat{r}$. The relevance of r is greater than or equal to $\hat{r}$. Let p be the position within $\hat{O}$ where r should be placed if selected. We differentiate between two cases.

i) All elements of $\hat{O}$ between $\hat{p}$ and p are greater than or equal to $\hat{r}$. Let O be the solution obtained from $\hat{O}$ by removing $\hat{r}$ and by inserting r. The solution O has a score greater than or equal to the one of $\hat{O}$ and uses an extra right-k-maximal element, thus contradicting the hypothesis.

ii) At least one element of $\hat{O}$ between $\hat{p}$ and p is strictly smaller than $\hat{r}$. Let $\hat{r}'$ be the rightmost element with this property. Let O be the solution obtained from $\hat{O}$ by removing $\hat{r}'$ and by inserting r. The solution O has a score strictly greater than $\hat{O}$, thus contradicting the optimality of $\hat{O}$. □

Algorithm 1 Right pruning

```
Input: A list $R$ of $n$ relevances and a size threshold $k$
Output: The list of the right-$k$-maximal elements of $R$
$result$ = List()
$heap$ = MinHeap()
for ($i = n$; $i > n$-$k$ and $i > 0$; $i$ -= 1) do
    $heap$.push($R[i]$)
    $result$.append($i$)
for ($i = n$-$k$; $i > 0$; $i$ -= 1) do
    if ($R[i] >$ $heap$.min()) then
        $heap$.replace_min($R[i]$)
        $result$.append($i$)
return $result$.reverse()
```

The right pruning can be computed in $\Theta(n \log k)$ time. Indeed, in this time we can compute all the right-k-maximal elements by employing a priority queue while scanning the list from right to left. At each step of the scanning, the data structure contains the top-k elements seen so far. If the current element enters the priority queue then it is right-k-maximal. This strategy is detailed in Algorithm 1.

Unfortunately, the right pruning does not guarantee an effective pruning of the original list. Indeed, there are lists where it does not reduce the number of elements to process with OPT. For example, when the elements of R are all distinct and sorted in decreasing order, all of them are right-k-maximal. We thus discuss how to improve the right pruning by using the next discretization step.

Discretization. This step aims at decreasing the number of elements selected by the right pruning in the worst case. This is done by creating a new list R_ϵ from R such that i) R_ϵ has a smaller number of distinct elements than R, and ii) the relevance of the optimal filtering of R_ϵ is guaranteed to be at least $(1-\epsilon)$ times the relevance of the optimal filtering of R. This way, the use of the previous right pruning is more effective in terms of removed elements still almost preserving the optimal solution.

The idea is to discretize the relevance of the elements of R to decrease the number of distinct elements. This discretization step trades-off between the approximation error of the solution and the obtained number of distinct elements. Let ϵ be the desired approximation error of ϵ-Filtering, with $0 < \epsilon < 1$. Let $r_{\min}$ and $r_{\max}$ be the minimum and maximum relevance of the elements in R, respectively. The idea of the discretization step is to partition the range $[r_{\min}, r_{\max}]$ into m intervals of elements. The elements belonging to the same interval are approximated in R_ϵ with the same relevance score, which equals the smallest relevance of the interval. The tricky part of this strategy is to decide how to partition the range $[r_{\min}, r_{\max}]$ to guarantee a ϵ approximation error and the minimum number of intervals m. Let $g(r)$ be the gain function used by the DCG metric, i.e., $g(r) = 2^r - 1$. We partition the range into intervals such that the ratio between the gains $g(\cdot)$ of the minimum and maximum relevances of each interval is at least (1-ϵ). We call ϵ−intervals the intervals in this partition. The following example shows how the discretization step works on the worst case list shown for the right pruning.

Example 5. Let R be the list $\langle 3, 2.99, 2.98, \ldots, 0.01\rangle$ of 300 elements and $\epsilon = 0.5$ the desired approximation error. Let $g(r)$ be the gain function used by the DCG metric, i.e., $g(r) = 2^r - 1$. The ϵ−intervals of R are the following ten intervals: $[0.01, 0.019)$, $[0.019, 0.039)$, $\ldots$, $[1.45, 2.17)$, $[2.17, 3.0]$. Within each interval the ratio between the gain of the maximum and the gain of the minimum is (1-ϵ). For example, for the last interval we have $g(2.17)/g(3) \approx 0.5$. The list R_ϵ is as follows $\langle 2.17, \ldots, 2.17, 1.45, \ldots, 1.45, \ldots, 0.01\rangle$.

The following lemma extends Lemma 4 to R_ϵ and shows that the right-k-maximal elements of R_ϵ can be exploited to find a (1-ϵ)-optimal filtering of R.

Lemma 6. *There exists an (1-ϵ)-optimal filtering of R composed of only right-k-maximal elements of R_ϵ.*

Proof. To prove the claim we need to show that the optimal filtering of R_ϵ is (1-ϵ)-optimal for R. Let O be an optimal filtering of R. Let $\hat{O}_\epsilon$ be the filtering built by selecting from R_ϵ the same elements composing O. The gain of each element of $\hat{O}_\epsilon$ is at least (1-ϵ) times the gain of its counterpart in O. Therefore, the relevance score of $\hat{O}_\epsilon$ is at least (1-ϵ) times the relevance score of O. In particular, every optimal solution O_ϵ of R_ϵ has a score greater than or equal to the relevance score of $\hat{O}_\epsilon$, therefore $Q(O_\epsilon) \geq (1\text{-}\epsilon)\, Q(O)$. The proof follows by applying Lemma 4 to R_ϵ and by showing that there exists a solution having score $Q(O_\epsilon)$ composed of only right-k-maximal elements of R_ϵ. □

The discretization of R into R_ϵ reduces the number of distinct elements from n to m. The following property provides an upper bound for m.

Property 7. The number of ϵ−intervals of R is

$$m \leq \left\lceil \log_{(1\text{-}\epsilon)} \left(g(r_{\min}) \,/ g(r_{\max}) \right) \right\rceil$$

Proof. Each ϵ−interval is such that the gain of the minimum element of the interval is (1-ϵ)-times the gain of its maximum. Therefore, by starting from the most relevant element of R, $r_{\max}$, the number of intervals m is the minimum value satisfying the following inequality: $g(r_{\max})\,(1\text{-}\epsilon)^m \leq g(r_{\min})$. □

The discretization step introduces an approximation error ϵ to reduce to m the number of distinct elements in the list R. This way, the combination with the right pruning guarantees that at most mk elements are selected.

However, the upper bound on m depends on both the minimum and the maximum relevances in R, namely, $r_{\min}$ and $r_{\max}$. If the gap between them is large, then m is large as well. In particular, it is possible to design an adversarial list R where the discretization is ineffective and the right pruning is forced to select all the elements. This is done by fixing a value of $r_{\min}$ and by choosing $r_{\max}$ to be large enough so that m equals n. Then, R is a decreasing list having an element for each ϵ−interval. This way, R_ϵ coincides with R and the pruning selects all its $m = n$ distinct elements.

Thresholding. This step works by removing those elements whose relevance is below a given threshold. In this way, we increase the value of $r_{\min}$ in the resulting list, referred as R^-, and, thus, we reduce the number m of possible ϵ−intervals. The threshold should be chosen so that i) the relevance of the optimal filtering of R^- is at least $(1-\epsilon)$ times the relevance of the optimal filtering of R, and ii) the value of m is guaranteed to be always much smaller than n. In this way, we can remove several elements with a small degradation of the solution as shown in the following example.

Example 8. Let R be the list $\langle 5, 0.1, \ldots, 0.1\rangle$ of 10 elements and $k = 10$. The optimal filtering selects the full list R, whose DCG is about 31.25. If we remove all elements having relevance 0.1 then the list R^- is $\langle 5\rangle$. The optimal filtering of R^- has a DCG score of 31, namely, it is a 0.99-approximation.

The following Lemma 9 says how to fix a threshold t to meet the desired approximation error ϵ.

LEMMA 9. *Let R^-_ϵ be the list obtained by removing the elements of R below the threshold $t = g^{-1}\left(\epsilon\, g(r_{max})/k\right)$ and then by discretizing the remaining elements using the ϵ−intervals. The optimal filtering of R^-_ϵ is a (1-ϵ)-approximation.*

PROOF. Let O and $\hat{O}$ be the optimal filterings of R and R^-_ϵ, respectively. The ratio ϵ/k is a value strictly smaller than 1, hence the threshold t is strictly smaller than $r_{\max}$. Let us consider the worst case list whose results, a part of the maximum, are infinitely close to the threshold t, but smaller than it, and are all placed on the right of $r_{\max}$. The solution $\hat{O}$ is thus formed by the singleton $\langle r_{\max}\rangle$ while the solution O is the list $\left\langle r_{\max}, \tilde{t}, \ldots, \tilde{t}\right\rangle$. Let $g(r)$ be the gain function used by the DCG metric, i.e., $g(r) = 2^r - 1$, and $d(r)$ be its discount function, i.e., $d(p) = 1/\log_2(p+1)$. The approximation factor achieved by using $\hat{O}$ is

$$(1-\epsilon) = \frac{Q(\hat{O})}{Q(O)} = \frac{g(r_{\max})\, d(1)}{g(r_{\max})\, d(1) + g(\tilde{t}) \sum_{i=2}^{k} d(i)},$$

which can be rewritten as

$$g(\tilde{t}) = \frac{\epsilon g(r_{\max})\, d(1)}{(1\text{-}\epsilon) \sum_{i=2}^{k} d(i)}.$$

The gain function $g()$ is a strictly increasing function, hence it admits an inverse function. Moreover, the discount function $d()$ is always smaller or equal than 1 and the factor (1-ϵ) is strictly smaller than 1, thus the denominator is strictly smaller than k. Therefore, the proof follows by using the previous rough approximations. □

The discretization and the thresholding steps can be combined together to effectively reduce the number of distinct values of the list R. Indeed, the threshold t limits the number of ϵ−intervals and thus the number of distinct values involved in the computation of the right-k-maximal elements. In particular, we can easily derive the following property by replacing $r_{\min}$ in Property 7 with the threshold t of Lemma 9.

PROPERTY 10. The number of ϵ−intervals of R^-_ϵ is

$$m \leq \left\lceil \log_{(1\text{-}\epsilon)}(\epsilon/k) \right\rceil.$$

The property above states that the maximum number of ϵ−intervals of R^-_ϵ can be upper bounded by a quantity that is independent from the length of R and its elements. As each right-k-maximal element r is followed by at most k elements having a relevance greater than or equal to r, the number of right-k-maximal elements of R^-_ϵ can be upper bounded by a quantity depending on ϵ and k only. Indeed, there are at most $km \leq k\lceil \log_{(1\text{-}\epsilon)}(\epsilon/k)\rceil$ right-k-maximal elements in R^-_ϵ.

Algorithm 2 ϵ-Pruning

```
Input: A list $R$ of $n$ relevances, a size threshold $k$ and a target
    approximation error $\epsilon$
Output: The list of the right-$k$-maximal elements of $R^-_\epsilon$
 1: $result$ = List()
 2: $heap$ = MinHeap()
 3: $cutoff$ = get_epsilon_cutoff($k$, $\epsilon$, max($R$))
 4: $i = n$
 5: for (; $heap$.size() < $k$ and $i > 0$; $i$ -= 1) do
 6:     if ($R[i] > cutoff$) then
 7:         $heap$.push($R[i]$)
 8:         $result$.append($i$)
 9: $intervals$ = get_eps_intervals($\epsilon$, max($R$), $heap$.min())
10: for ($cur$ = 1; $i > 0$; $i$ -= 1) do
11:     if ($R[i] > intervals[cur]$) then
12:         $heap$.replace_min($R[i]$)
13:         $result$.append($i$)
14:         while ($intervals[cur] < heap$.min()) $cur$ += 1
15: return $result$.reverse()
```

We refer to the strategy that finds the right-k-maximal elements of R^-_ϵ as ϵ-Pruning. An efficient implementation of ϵ-Pruning is reported in Algorithm 2. The proposed algorithm computes the threshold t using the formula of Lemma 9 (line 3) and the ϵ−intervals bounds (line 9). The algorithm scans the list R and uses a priority queue to keep track of the top-k right-k-maximal elements of R^-_ϵ seen so far. We use a cursor *cur* to keep track of the ϵ−interval of the smallest element of the priority queue. If the current element fits into a higher ϵ−interval then it is right-k-maximal for R^-_ϵ and it enters into the priority queue (line 11). Here, we do not discretize all elements of R, because it would require to find the corresponding ϵ−interval of each element even if it is not necessary. Instead, we use the cursor *cur* to find the corresponding ϵ−interval of only the elements entering into the priority queue. Since the cursor just advances in the *intervals* array (line 14), the number of times we update the cursor position is independent of the length of the list and it is limited by the number of intervals, i.e., $\Theta(m)$.

Since the maximum number of right-k-maximal elements of R^-_ϵ is km, Algorithm 2 costs $\Theta(n + km \log k)$ time. The first factor is due to the cost of a linear scan of the list R, while the second factor is due to the cost of $\Theta(km)$ updates of a priority queue of size k. Therefore, ϵ-Pruning is an efficient and effective way for reducing the number of elements to be processed by OPT.

ϵ−Filtering. Our algorithm is then the combination of ϵ-Pruning and OPT. Theorem 11 shows that ϵ-Filtering finds a (1-ϵ)-optimal solution and trades-off between effectiveness and efficiency through the approximation error ϵ.

THEOREM 11. ϵ-Filtering finds a (1-ϵ)-optimal filtering in $\Theta\left(n + k^2 \log_{(1\text{-}\epsilon)}(\epsilon/k)\right)$ time.

PROOF. ϵ-Filtering is composed of two phases: ϵ-Pruning and OPT. ϵ-Pruning finds the right-k-maximal elements of R^-_ϵ, which, according to Lemma 4, contain the optimal solution of the approximate list. OPT finds the optimal solution of the right-k-maximal elements of R^-_ϵ, which is (1-ϵ)-optimal according to Lemma 9.

The time complexity of ϵ-Pruning is $\Theta(n + km \log k)$ while the one of OPT is $\Theta(k^2 m)$ when applied to the $\Theta(km)$ elements selected by ϵ-Pruning. The claim follows from Property 10. □

ϵ-Filtering solves the relevance-aware filtering problem by combining three steps, i.e., right pruning, discretization, and thresholding, with OPT. ϵ-Filtering provides strong guarantees on the effectiveness of the filtered list and trades efficiency for effectiveness driven by the approximation error ϵ.

Distributed setting. Real-world search services exploit distributed query processing architectures [2, 12] to deliver content to their users. Typically, a distributed environment consist of a set of indices each one queried by a search engine. On top of them, a meta-search engine distributes the query to all search engines. Each of them query its index in parallel and produce a ranked list of top-k results that is sent back to the meta-search engine. Finally, the meta-search engine aggregates the results and send them to the user.

OPT cannot be applied in a distributed query processing architecture since it computes the optimal solution by taking into account all the results available. One possibility to overcome this issue is to deploy OPT in the meta-search engine. However, sending all the results matching a query from a search engine to the meta-search engine is unfeasible. This would lead to high network communication overhead between machines. Moreover, by applying OPT on partial lists of results, i.e., on a search engine level, the final solution is not guaranteed to be optimal.

ϵ-Filtering can be applied in a distributed query processing architecture as it preserves the approximation guarantees. The application of ϵ-Filtering to a distributed scenario requires: i) each search engine to apply ϵ-Pruning to the local results and to send the right-k-maximal elements to the meta-search engine, ii) the meta-search engine to merge the lists received by preserving the per-attribute sorting and to apply ϵ-Filtering to the merged list. By exploiting Property 10, we can prove that the number of elements transferred by each search engine is at most $k\lceil \log_{(1\text{-}\epsilon)}(\epsilon/k)\rceil$ thus confirming the feasibility of the approach.

4 EXPERIMENTS

We evaluate the performance of ϵ-Filtering and its competitors on two real datasets, namely GoogleLocalRec and AmazonRel. The two datasets are built around two different real-world use cases and by exploiting state-of-the-art relevance estimators. Moreover, they present a high number of results per query thus allowing us to perform a comprehensive assessment of the efficiency of the filtering methods.

The GoogleLocalRec dataset is built by using the Google Local data and a state-of-the-art recommender system (TransFM) that recently achieved the best performance in the task of sequential recommendation [11], i.e., the task of predicting users' future behaviour given their historical interactions. The GoogleLocalRec dataset is made up of a large collection of temporal and geographical information of businesses and reviews [5]. We used the GoogleLocalRec dataset to produce, given a user, a list of relevant businesses to recommend to her. In particular, we focused on the city of New York, which is the city with the highest density of data. Therefore, we preprocessed the dataset by following the methodology described in the original paper [11] and we trained a recommendation model (TransFM$_{\text{content}}$) using the reviews of the businesses within a radius of 50Km from the center of New York. Then, we randomly selected $10,000$ test users. For each test user, we sorted the businesses in New York by distance and we estimated their relevance by predicting the recommendation score with the TransFM$_{\text{content}}$ model. The GoogleLocalRec dataset we built is thus composed of $10,000$ users, also referred to as queries. Each user comes with a list of exactly $16,000$ recommendations.

The AmazonRel dataset is built by using one of the winning solutions[1] of the Crowdflower Search Results Relevance[2] Kaggle competition, whose goal was to create an open-source model to help small e-commerce sites to measure the relevance of the search results they provide. However, the test set distributed as part of the Kaggle competition contains only few results associated with each query. Therefore, we extended the test set using the Amazon dataset introduced by [10], whose data come from the Amazon e-commerce website and span from May 1996 to July 2014. We added to the original result list of each query all Amazon items having at least one of the query terms in the title or in the description. Then, we sorted the results by price and we computed their relevance by using the winning solution of the Crowdflower Search Results Relevance competition. The resulting AmazonRel dataset is composed of 260 queries. We then removed 10 queries characterized by less than 500 results. The resulting dataset is made up of 250 queries and each query comes with $100,000$ results on average.

Metrics. We assess the performance of ϵ-Filtering and its competitors by using two different metrics.

- DCG $(\langle r_1, ..., r_n\rangle) = \sum_{p=1}^{n} \frac{2^{r_p}-1}{\log_2(p+1)}$, which actively rewards relevant results. It is composed of an exponential gain function for relevance, $g(r) = 2^r - 1$, combined with a smooth discount factor for position, $d(p) = 1/\log_2(p+1)$.
- DCG-LZ $(\langle r_1, ..., r_n\rangle) = \sum_{p=1}^{n} \frac{r_p}{p}$, which penalizes the results that are not in the top positions. It combines a linear gain function for relevance, $g(r) = r$, with the Zipfian discount factor for position [8], $d(p) = 1/p$.

We employed the two aforementioned metrics because they are widely used within the IR community [6, 8, 16]. Moreover, due to their differences in $g(\cdot)$ and $d(\cdot)$, they provide a very different evaluation: DCG actively promotes the relevance, while DCG-LZ actively discounts the results that are in the lower positions of the ranked list.

Filtering methods. We compare the performance of ϵ-Filtering against three strong competitors introduced by Spirin *et al.* [14]: OPT, Top$k$-OPT, and Cutoff-OPT. The relevance threshold used by Cutoff-OPT is query-based and it is defined as $(r_{\max} + r_{\min})/2$ where $r_{\max}$ ($r_{\min}$) is the maximum (minimum) relevance of the list.

[1] https://github.com/geffy/kaggle-crowdflower

[2] https://www.kaggle.com/c/crowdflower-search-relevance

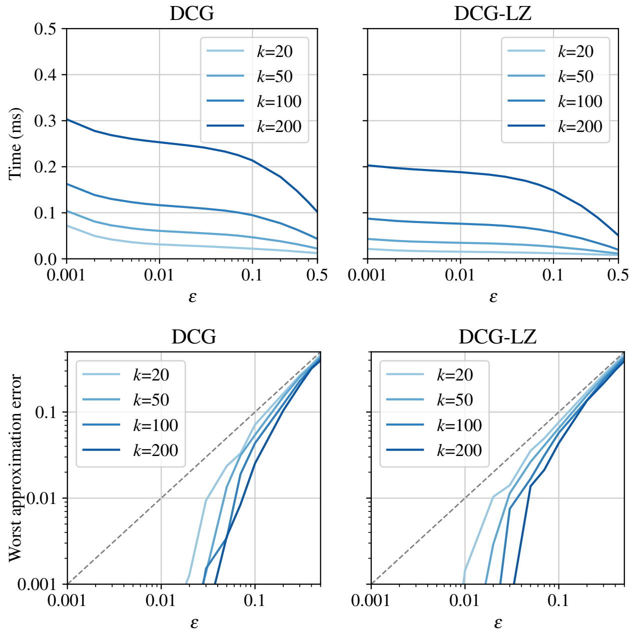

Figure 1: Average filtering time (top) and worst approximation error (bottom) of ϵ-Filtering by varying ϵ and k.

Testing details. All algorithms are implemented in C++11 and compiled with GCC 5.4.0 using the highest optimization settings. The tests were performed on a machine with eight Intel Core i7-7700 cores clocked at 3.60GHz, 64GiB RAM. We report the results of each method as the average execution time (in milliseconds) of five independent runs.

Reproducibility. To ease the reproducibility of the results we release the GoogleLocalRec and AmazonRel datasets along with our implementation of OPT, Top$k$-OPT, Cutoff-OPT, and ϵ-Filtering[3].

4.1 Assessment of ϵ-Filtering

We now present an analysis of the performance of ϵ-Filtering as a function of several parameters, i.e., ϵ, k, and n. Due to space limitation, we report these experiments on the GoogleLocalRec dataset only. We report the average performance achieved on the 10, 000 test queries.

Assessment by varying ϵ and k. The parameter ϵ drives the efficiency of ϵ-Filtering through the granularity of the approximation. As shown in Section 3.2, it guarantees the desired approximation error by controlling the width of the ϵ−intervals. Figure 1 (top) shows the average filtering time achieved by ϵ-Filtering by varying ϵ (x-axis) from 0.001 to 0.5. Each line refers to a different value of $k \in \{20, 50, 100, 200\}$. As expected, the filtering time increases when decreasing the desired approximation error ϵ. The reported results are always below 0.3 milliseconds. Specifically, the filtering time ranges from 0.1ms to 0.3ms when $k = 200$ and it softly increases with values of ϵ smaller than 0.05 for all values of k tested. This ensures that, even when decreasing the desired approximation error below 0.05, ϵ-Filtering is a feasible solution for online applications characterized by tight time constraints.

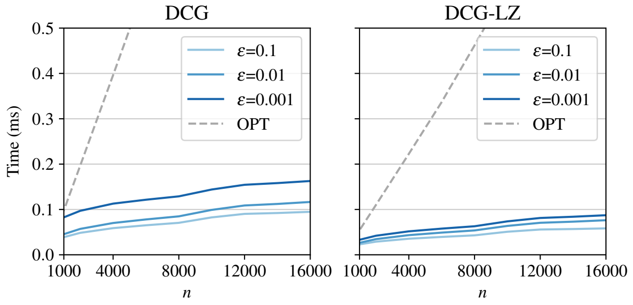

Figure 2: Average filtering time of ϵ-Filtering and OPT by varying n and ϵ.

Figure 1 (top) also shows that the parameter k impacts the efficiency more than the approximation error ϵ. The trend is clear when increasing k from 100 to 200: the difference in time is greater than the one achieved by varying ϵ. This is due to the fact that the parameter k appears as a quadratic factor in the formula defining the cost of ϵ-Filtering (see Theorem 11).

Figure 1 (bottom) reports the worst, i.e., maximum, approximation error achieved on all test queries by varying the parameters ϵ and k. Indeed, this experiment helps to understand what is the real approximation achieved by the filtering. The gray dashed line reports the theoretical approximation error that is represented by those points whose approximation is exactly ϵ. The experimental worst approximation error sharply decreases when decreasing ϵ. The achieved error is always far from the theoretical approximation error. This result confirms that ϵ-Filtering achieves in practice very good approximations of the optimal solution.

Assessment by varying n. We now evaluate the performance of ϵ-Filtering and OPT by varying n: for each $n' < n$ we evaluate all strategies on the first n' results sorted by distance of each test query. The result of this analysis is reported in Figure 2. We perform the experiment by setting $k = 100$. Results show the average time needed by ϵ-Filtering and OPT to process the 10, 000 test queries. The time required by OPT to filter the lists grows sharply when increasing n, while the time required by ϵ-Filtering on the same lists increases slowly. The time required by the latter doubles from 1, 000 to 16, 000 results, while the time required by the former increases by 17 times in the same range. It is worth highlighting that a remarkable property of ϵ-Filtering is that the difference in time between different values of ϵ is almost constant when increasing n. Indeed, given an approximation error ϵ, the average filtering time of ϵ-Filtering is marginally affected by the size of the list. Therefore, ϵ-Filtering is a feasible solution for relevance-aware filtering when dealing with very long lists of results.

Speedup. The parameter ϵ affects the number of elements prefiltered by ϵ-Pruning which in turn must be processed by OPT. Figure 3 (top) shows the fraction of elements removed by ϵ-Pruning on average. As expected, the number of elements it discards grows

[3] https://github.com/hpclab/fast-approximate-filtering

Table 1: Average filtering time (above) and worst approximation error (below) achieved by OPT, Cutoff-OPT, Top$k$-OPT and ϵ-Filtering with $n = 16,000$ by varying k. Time is reported in milliseconds. The best results are in bold.

Strategy	**GoogleLocalRec** by varying k				**AmazonRel** by varying k			
	20	50	100	200	20	50	100	200
OPT	0.206	0.540	1.019	1.991	0.198	0.537	1.010	1.976
Cutoff-OPT	0.156 (1×)	0.353 (2×)	0.670 (2×)	1.281 (2×)	0.199 (1×)	0.454 (1×)	0.868 (1×)	1.686 (1×)
(*no guarantees*)	0.20	0.29	0.34	0.38	0.23	0.33	0.39	0.43
Top$k$-OPT	0.017 (12×)	**0.022 (25×)**	**0.032 (32×)**	**0.055 (37×)**	0.018 (11×)	0.025 (21×)	0.039 (26×)	**0.067 (30×)**
(0.5-*approximation*)	0.15	0.14	0.13	0.11	0.14	0.14	0.13	0.13
ϵ-Filtering	**0.012 (17×)**	0.026 (21×)	0.058 (17×)	0.149 (13×)	**0.011 (19×)**	**0.018 (29×)**	**0.037 (27×)**	0.094 (21×)
(ϵ=0.1)	0.08	0.07	0.06	0.04	0.06	0.05	0.05	0.04
ϵ-Filtering	0.015 (13×)	0.035 (16×)	0.076 (13×)	0.188 (11×)	0.013 (16×)	0.025 (22×)	0.054 (19×)	0.138 (14×)
(ϵ=0.01)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
ϵ-Filtering	0.021 (10×)	0.043 (13×)	0.087 (12×)	0.203 (10×)	0.016 (13×)	0.030 (18×)	0.062 (16×)	0.151 (13×)
(ϵ=0.001)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

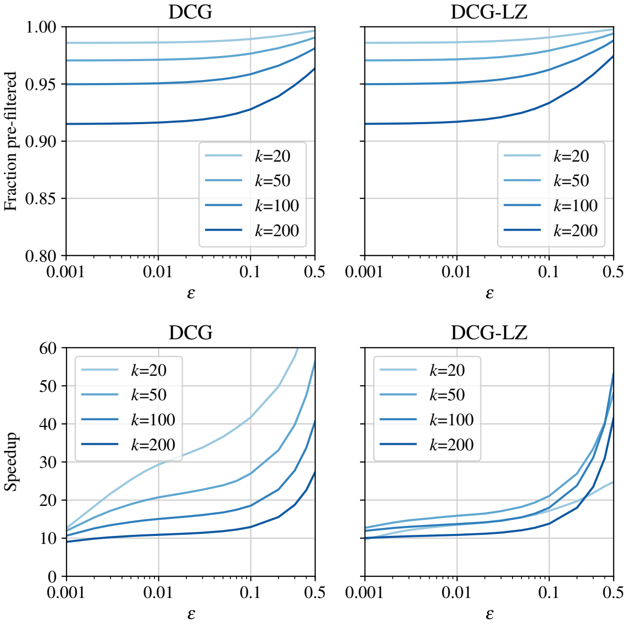

Figure 3: Average fraction of pre-filtered elements (top) and average speedup (bottom) of ϵ-Filtering by varying ϵ and k.

when decreasing the approximation error ϵ and, specifically, it is almost flat when ϵ is smaller than 0.01. In particular, with k up to 100, our strategy pre-filters more than 95% of the elements on average. This important reduction of the original list to less than 5% of the elements boosts the filtering performance by reducing the amount of work done by OPT to filter the list.

Figure 3 (bottom) shows the average speedup of ϵ-Filtering against OPT measured on all the test queries. Results show a speedup of more than one order of magnitude for almost all combinations of parameters tested. Specifically, with $\epsilon = 0.01$ and k up to 100, our strategy is from 15 to 30 times faster than OPT when using the DCG metric, and from 14 to 16 times faster when using the DCG-LZ metric. It is worth highlighting that the speedups achieved by using the DCG-LZ metric are smaller than the ones achieved by the DCG metric because the former is computationally less expensive to compute. When increasing ϵ to 0.05, with $k = 20$, ϵ-Filtering is 37 and 15 times faster than OPT using both DCG and DCG-LZ, respectively. When further increasing ϵ to 0.1, the speedup is even bigger and improves the previous results of about 30%.

4.2 Experimental Comparisons

We now assess the performance of ϵ-Filtering against the state-of-the-art competitors introduced by [14], i.e., OPT, Top$k$-OPT and Cutoff-OPT, by setting $n = 16,000$, i.e., the maximum length of the results lists available in the GoogleLocalRec dataset. We perform the same experiment on the AmazonRel dataset by employing, for each query, the first 16,000 results sorted by price. In the latter dataset, we discard the queries with less than 16,000 results. The results of the experiment are summarized in Table 1, where, by varying $k \in \{0.1, 0.01, 0.001\}$, for each method we report i) the average filtering time, ii) the speedup achieved over OPT, and iii) the worst approximation error achieved on all queries. The results reported refer to the DCG-LZ metric. The performance of OPT ranges from about 0.2ms ($k = 20$) to about 2ms ($k = 200$). Cutoff-OPT and Top$k$-OPT achieve a better performance than the one of OPT. The speedup of Cutoff-OPT ranges from 1× to 2×. The best performance in terms of average filtering time is achieved by Top$k$-OPT with a speedup ranging from 25× to 37× when $k \in \{50, 100, 200\}$ on the GoogleLocalRec dataset. Furthermore, Top$k$-OPT achieves the best performance with a speedup of 30× when $k = 200$ on the AmazonRel dataset.

It is worth reminding that Cutoff-OPT has no proven approximation guarantees while Top$k$-OPT is 0.5−optimal. To assess the actual performance, we experimentally evaluate the worst approximation error achieved by the two heuristics. We observe that the performance of Cutoff-OPT ranges from 0.2 ($k = 20$) to 0.38 ($k = 200$) on the GoogleLocalRec dataset while it scores from 0.23 ($k = 20$) to 0.43 ($k = 200$) on the AmazonRel dataset. Top$k$-OPT shows better

Table 2: Average filtering time (above) and worst approximation error (below) achieved by OPT, Cutoff-OPT, Top$k$-OPT and ϵ-Filtering with $k = 100$ and by varying n. Time is reported in milliseconds. The best results are in bold.

Strategy	**AmazonRel** by varying n			
	50, 000	100, 000	200, 000	500, 000
OPT	5.003	9.809	19.280	47.536
Cutoff-OPT (*no guarantees*)	2.255 (2×)	4.660 (2×)	8.863 (2×)	8.387 (6×)
	0.20	0.04	0.02	0.00
Top$k$-OPT (0.5-*approximation*)	0.073 (68×)	0.119 (82×)	0.209 (92×)	0.474 (100×)
	0.11	0.10	0.11	0.05
ϵ-Filtering (ϵ=0.1)	**0.059 (85×)**	**0.086 (114×)**	**0.138 (139×)**	**0.292 (163×)**
	0.05	0.05	0.05	0.03
ϵ-Filtering (ϵ=0.01)	0.082 (61×)	0.112 (87×)	0.166 (116×)	0.324 (147×)
	0.00	0.00	0.00	0.00
ϵ-Filtering (ϵ=0.001)	0.091 (55×)	0.124 (79×)	0.179 (108×)	0.336 (141×)
	0.00	0.00	0.00	0.00

performance. The worst approximation error of the latter heuristic ranges from 0.15 ($k = 20$) to 0.11 ($k = 200$) on the GoogleLocalRec dataset while on the AmazonRel dataset the difference is smaller as it scores from 0.14 ($k = 20$) to 0.13 ($k = 200$). The results in terms of worst approximation error show that the two heuristics are far from being optimal.

The performance of ϵ-Filtering in terms of speedup w.r.t. OPT ranges from 10× to 29×. As expected, the best speedup is achieved when $\epsilon = 0.1$. When decreasing the approximation error ϵ, the speedup reduces as ϵ-Filtering takes more time to compute a more accurate solution. Nevertheless, ϵ-Filtering achieves noteworthy speedups, from 10× to 18×, even when employing $\epsilon = 0.001$, i.e., when we require a 1‰ approximation error. In terms of worst approximation error, ϵ-Filtering achieves results ranging from 0.08 to 0. In details, when $\epsilon = 0.1$ the worst approximation error measured ranges from 0.08 to 0.04 for the GoogleLocalRec dataset, while it ranges from 0.06 to 0.04 for the AmazonRel dataset. The result highlights superior performance of ϵ-Filtering w.r.t Top$k$-OPT and Cutoff-OPT. Indeed, when $\epsilon \in \{0.01, 0.001\}$, the worst approximation error achieved by ϵ-Filtering is 0, i.e., it is always able to compute the optimal solution for all the queries of the two datasets while achieving speedups ranging from 10× to 29× over OPT. Therefore, ϵ-Filtering is able to compute solutions that are either optimal or very close to the optimal with a significant reduction of the filtering time.

Since the AmazonRel dataset is made up of longer lists, i.e., up to 600, 000 results per query, we now analyze the performance of the different methods by varying n. The results reported in Table 2 refers to the DCG-LZ metric. We perform the analysis by varying n: for each $n' < n$ we evaluate all strategies on the first n' results sorted by price. We discard the queries with less than n' results. We perform the experiment by setting $k = 100$. We observe that, by increasing n, the performance of OPT significantly degrades. When dealing with lists of 500, 000 items, OPT needs 48ms to compute the solution, while when $n = 100, 000$ it needs about 10ms. This confirms that OPT is not always a feasible choice. The Cutoff-OPT heuristic shows limited speedups compared to the ones of the heuristic Top$k$-OPT. Indeed, the speedup achieved by Top$k$-OPT improves by increasing the value of n. It is particularly fast even with very long lists ($n = 500, 000$) achieving a speedup of 100× over OPT. However, the results show that, by increasing n, the time spent by Top$k$-OPT grows faster than the one of ϵ-Filtering. This result confirms the theoretical analysis we provide in Section 3, where we show that the time complexity of Top$k$-OPT, i.e., $\Theta\left(n \log k + k^2\right)$, has a factor $\log k$ more than the one of ϵ-Filtering, which is $\Theta\left(n + k^2 \log_{(1\text{-}\epsilon)} \frac{\epsilon}{k}\right)$.

ϵ-Filtering is the best performing method for all values of n when $\epsilon = 0.1$. In these settings, ϵ-Filtering is remarkably faster than the two heuristics. The best speedup, i.e., 163×, is achieved with lists of 500, 000 items. When decreasing ϵ to lower values, i.e., 0.01 and 0.001, the performance of ϵ-Filtering does not degrade significantly achieving a maximum speedup of 141× in the latter case. Moreover, when $\epsilon \in \{0.01, 0.001\}$, the worst approximation error achieved is always zero meaning that ϵ-Filtering always found an optimal solution for all tested queries.

5 CONCLUSIONS AND FUTURE WORK

We proposed ϵ-Filtering, an efficient approximate algorithm for solving the relevance-aware filtering problem with strong approximation guarantees on the relevance of the final list. Given an allowed approximation error ϵ, the proposed algorithm finds a (1-ϵ)–optimal filtering, i.e., the relevance of its solution is at least (1-ϵ) times the optimum. We proposed a comprehensive evaluation of ϵ-Filtering against three state-of-the-art competitors, i.e., OPT, Top$k$-OPT, and Cutoff-OPT, on two real-world public datasets. Experiments show that ϵ-Filtering achieves the desired levels of effectiveness with a speedup of up to two orders of magnitude with respect to the optimal solution, OPT, while guaranteeing very small approximation errors. Indeed, experiments shows that, when decreasing ϵ to small values, e.g., 0.01 and 0.001, the worst approximation error achieved is always zero meaning that ϵ-Filtering always finds the optimal solution on all tested queries while achieving speedups ranging from 10× to 147× over OPT.

As future work, we plan to investigate more restrictive classes of right-k-maximal elements. Indeed, the right-k-maximal elements assume a central role in our work and the discovery of new classes of these elements that can be efficiently identified could lead to even faster approximate algorithms. We also plan to evaluate ϵ-Filtering in a real-world distributed query processor to comprehensively assess its performance in this scenario.

6 ACKNOWLEDGEMENTS

This paper is partially supported by the BIGDATAGRAPES (EU H2020 RIA, grant agreement №780751), the "Algorithms, Data Structures and Combinatorics for Machine Learning" (MIUR-PRIN 2017), and the OK-INSAID (MIUR-PON 2018, grant agreement №ARS01_00917) projects.

REFERENCES

[1] Ricardo A. Baeza-Yates and Berthier A. Ribeiro-Neto. 2011. *Modern Information Retrieval - the concepts and technology behind search, Second edition.* Pearson Education Ltd., Harlow, England. http://www.mir2ed.org/

[2] Berkant Barla Cambazoglu and Ricardo A. Baeza-Yates. 2011. Scalability Challenges in Web Search Engines. In *Advanced Topics in Information Retrieval*, Massimo Melucci and Ricardo A. Baeza-Yates (Eds.). The Information Retrieval Series, Vol. 33. Springer, 27–50. DOI: http://dx.doi.org/10.1007/978-3-642-20946-8_2

[3] David Carmel, Liane Lewin-Eytan, Alex Libov, Yoelle Maarek, and Ariel Raviv. 2017. Promoting Relevant Results in Time-Ranked Mail Search. In *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*, Rick Barrett, Rick Cummings, Eugene Agichtein, and Evgeniy Gabrilovich (Eds.). ACM, 1551–1559. DOI: http://dx.doi.org/10.1145/3038912.3052659

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms, 3rd Edition.* MIT Press. http://mitpress.mit.edu/books/introduction-algorithms

[5] Ruining He, Wang-Cheng Kang, and Julian McAuley. 2017. Translation-based Recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems, RecSys 2017, Como, Italy, August 27-31, 2017*, Paolo Cremonesi, Francesco Ricci, Shlomo Berkovsky, and Alexander Tuzhilin (Eds.). ACM, 161–169. DOI: http://dx.doi.org/10.1145/3109859.3109882

[6] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)* 20, 4 (2002), 422–446. DOI: http://dx.doi.org/10.1145/582415.582418

[7] Myeongjae Jeon, Saehoon Kim, Seung-won Hwang, Yuxiong He, Sameh Elnikety, Alan L. Cox, and Scott Rixner. 2014. Predictive parallelization: taming tail latencies in web search. In *The 37th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '14, Gold Coast , QLD, Australia - July 06 - 11, 2014*, Shlomo Geva, Andrew Trotman, Peter Bruza, Charles L. A. Clarke, and Kalervo Järvelin (Eds.). ACM, 253–262. DOI: http://dx.doi.org/10.1145/2600428.2609572

[8] Evangelos Kanoulas and Javed A. Aslam. 2009. Empirical justification of the gain and discount function for nDCG. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management, CIKM 2009, Hong Kong, China, November 2-6, 2009.* 611–620. DOI: http://dx.doi.org/10.1145/1645953.1646032

[9] Ron Kohavi, Alex Deng, Brian Frasca, Toby Walker, Ya Xu, and Nils Pohlmann. 2013. Online controlled experiments at large scale. In *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013*, Inderjit S. Dhillon, Yehuda Koren, Rayid Ghani, Ted E. Senator, Paul Bradley, Rajesh Parekh, Jingrui He, Robert L. Grossman, and Ramasamy Uthurusamy (Eds.). ACM, 1168–1176. DOI: http://dx.doi.org/10.1145/2487575.2488217

[10] Julian J. McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. 2015. Image-Based Recommendations on Styles and Substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, Santiago, Chile, August 9-13, 2015*, Ricardo A. Baeza-Yates, Mounia Lalmas, Alistair Moffat, and Berthier A. Ribeiro-Neto (Eds.). ACM, 43–52. DOI: http://dx.doi.org/10.1145/2766462.2767755

[11] Rajiv Pasricha and Julian McAuley. 2018. Translation-based factorization machines for sequential recommendation. In *Proceedings of the 12th ACM Conference on Recommender Systems, RecSys 2018, Vancouver, BC, Canada, October 2-7, 2018*, Sole Pera, Michael D. Ekstrand, Xavier Amatriain, and John O'Donovan (Eds.). ACM, 63–71. DOI: http://dx.doi.org/10.1145/3240323.3240356

[12] Mr Biraj Patel and Dr Dipti Shah. 2011. Meta search ranking strategies. *International Journal of Information and Computing Technology (RESEARCH@ ICT) ISSN* 976, 5999 (2011), 24–25.

[13] Tao Qin, Tie-Yan Liu, Jun Xu, and Hang Li. 2010. LETOR: A benchmark collection for research on learning to rank for information retrieval. *Inf. Retr.* 13, 4 (2010), 346–374. DOI: http://dx.doi.org/10.1007/s10791-009-9123-y

[14] Nikita V. Spirin, Mikhail P. Kuznetsov, Julia Kiseleva, Yaroslav V. Spirin, and Pavel A. Izhutov. 2015. Relevance-aware Filtering of Tuples Sorted by an Attribute Value via Direct Optimization of Search Quality Metrics. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, Santiago, Chile, August 9-13, 2015*, Ricardo A. Baeza-Yates, Mounia Lalmas, Alistair Moffat, and Berthier A. Ribeiro-Neto (Eds.). ACM, 979–982. DOI: http://dx.doi.org/10.1145/2766462.2767822

[15] Andrew Trotman, Surya Kallumadi, and Jon Degenhardt. 2018. High Accuracy Recall Task. In *The SIGIR 2018 Workshop On eCommerce co-located with the 41st International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2018), Ann Arbor, Michigan, USA, July 12, 2018. (CEUR Workshop Proceedings)*, Jon Degenhardt, Giuseppe Di Fabbrizio, Surya Kallumadi, Mohit Kumar, Andrew Trotman, Yiu-Chang Lin, and Huasha Zhao (Eds.), Vol. 2319. CEUR-WS.org. http://ceur-ws.org/Vol-2319/paper21.pdf

[16] Yining Wang, Liwei Wang, Yuanzhi Li, Di He, and Tie-Yan Liu. 2013. A Theoretical Analysis of NDCG Type Ranking Measures. In *COLT 2013 - The 26th Annual Conference on Learning Theory, June 12-14, 2013, Princeton University, NJ, USA (JMLR Workshop and Conference Proceedings)*, Shai Shalev-Shwartz and Ingo Steinwart (Eds.), Vol. 30. JMLR.org, 25–54. http://jmlr.org/proceedings/papers/v30/Wang13.html