Profile Diversity for Query Processing using User Recommendations

Maximilien Servajean, Reza Akbarinia, Esther Pacitti, Sihem Amer-Yahia

Profile Diversity for Query Processing using User Recommendations[☆]

Maximilien Servajean[a,*], Reza Akbarinia[a], Esther Pacitti[a], Sihem Amer-Yahia[b]

[a]*INRIA & LIRMM, University of Montpellier France*
[b]*CNRS, LIG*

Abstract

More than 90% of the queries submitted to content sharing platforms, such as *Flickr*, are vague, *i.e.* only contain a few keywords, thus complicating the task of effectively returning interesting results. To overcome this limitation, many platforms use recommendation strategies to filter the results. But, recommendations usually tend to return highly redundant items. Content diversification has been studied as a solution to overcome this problem. However, it may suffer from at least two limitations: poor content description and semantic ambiguity.

In this paper, we investigate profile diversity for searching web items. Profile diversification enables to address the problem of returning redundant items, and enhances the quality of diversification. We propose a threshold-based approach to return the most relevant and most popular documents while satisfying content and profile diversity constraints. Our approach includes a family of techniques allowing to efficiently retrieve the desired documents. To evaluate our solution, we have run intensive experiments, including a user survey, on three datasets; in more than 75% of the cases, profile diversity is similar or preferred by the users compared to other approaches. Additionally our optimization techniques enable to reduce the response time up to 12 times compared to a baseline greedy diversification algorithm.

Keywords: search and recommendation, diversity, top-k

1. Introduction

With the advent of Web 2.0, a large number of content sharing platforms, such as *Delicious*[1] and *Flickr*[2], have emerged. Users have become massive producers of content such as images, videos or even bookmarks and can express their opinions on these content by writing comments or submitting tags. Unfortunately, this overwhelming amount of content complicates the process of retrieving relevant data items. In 2009, after analyzing 10 million *Yahoo! Travel* queries, Yu et al. [1] showed that more than 90% of these queries are generic. A generic query contains only very few keywords such as "family trip". The description offered

[☆]Work conducted within the Institut de Biologie Computationnelle and partially funded by the labex NUMEV and the CNRS project Mastodons.
[*]Corresponding author, *Phone number*: +33 6 13 91 15 64
Email address: `servajean@lirmm.fr` (Maximilien Servajean)
[1]`www.delicious.com`
[2]`www.flickr.com`

by these queries is so vague that it is not possible to effectively return interesting results to the users. To overcome this limitation, many platforms such as *Amazon*[3] or *Netflix*[4] use recommendation strategies to filter the results based on each user's profile, thus improving user satisfaction.

Although the recommendations enable a huge filtering of the global corpus based on the user profiles, they usually tend to return highly redundant items (*i.e.* overspecialized items). Redundancy may become very annoying to the users [1], thus leading to a loss of attendance of the platform. For instance, on *Delicious*, the tag (*i.e.* keyword) `#tunisia`, only returns content about the post-revolution difficulties of the country. Content diversification has been studied as a solution to overcome this problem of returning highly relevant but too redundant data items. The intuition is to return items that are not only relevant with respect to a query q or to the user's profile, but also different from each other. Much promising results have been produced using content diversification [2, 1, 3].

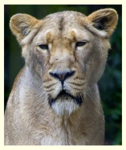
However, content diversification remains limited in some cases. Table 1 presents four use cases extracted from our experiments using a dataset from *Flickr*. Two users have submitted the query "lion". Cases 1 and 2 show the recommended pictures for each user using no diversity and content diversity respectively. Cases 3 and 4 show the recommended pictures for the same two users using profile diversity. Notice that for Case 1 there are only pictures of lions in similar scenarios (*i.e.* zoo and safari) while for Case 2 pictures are more diverse (*i.e.* a Chinese statue of a lion). Nevertheless, in Case 2 we can still observe several pictures that appear in Case 1. Two reasons can be highlighted to explain the low diversity of Case 2:

1. **Poor content description:** in several platforms such as *Delicious*, the low quality of content descriptions reduces the effectiveness of content diversification [1]. For instance, two pictures of the Eiffel Tower tagged respectively with "Paris trip holiday" and "Eiffel Tower Paris" may be considered as diverse, while they are very redundant.
2. **Semantic ambiguity:** when searching documents, a particular word can refer to several subjects. For instance, the query `diversity` may refer to *bio-diversity*, *information retrieval diversity* and much more. Thus, with content diversification, the diversified recommended items could still refer to the same semantic.

With profile diversification, the recommendation system tries to take into account the recommendations of users with diversified profiles. It allows to reduce the limitations due to poor content description and to increase the probability of recommending items corresponding to different topics. To illustrate the benefits of profile diversification, we briefly discuss the use-case depicted in Table 1. Notice that, the recommended pictures for cases 3 and 4, in addition to be relevant with respect to the query "lion", and to the users 1 and 2's profiles, also take into account the diversity of the pictures and of the profiles of the users sharing relevant pictures, thus providing a higher quality of diversification compared to the first two cases.

[3]`www.amazon.com`
[4]`www.netflix.com`

Table 1: Case study of content and profile diversity for the query “lion”.

Methods	Case 1 No Diversity	Case 2 Content Diversity	Case 3 Profile Diversity 1	Case 4 Profile Diversity 2
Users	User 1	User 2	User 1	User 2
Profiles	Sea, boat, fishing	Savanna, jungle, Africa	Sea, boat, fishing	Savanna, jungle, Africa
Results				
				

In [4], a diversified recommendation method is proposed, that takes into account the diversity of comments and users meta-data. In [1], diversification is done in the sense that given the *users* $\times$ *items* matrix, and a user u, each recommended item to u should be provided by different users.

In this paper, we propose an original approach for profile diversification [5] exploiting a probabilistic model. To process profile diversification queries in large-scale datasets, new efficient techniques are required. Our probabilistic diversification approach relies on *top-k* processing over inverted lists. An inverted list is associated to a specific keyword and items in it are sorted in decreasing order of relevance with respect to the selected word. However, since our model takes into account the diversity, an item's score depends not only on its relevance to the query, but also to the items previously added to the results list, thus the classical *top-k* algorithms are no longer suitable. Therefore, we developed new techniques to compute the diversified *top-k* results. In these techniques, we optimized the number of accesses to the sorted lists and the number of diverse items considered during query processing. We also proposed a model for profile diversification that can be computed efficiently.

In summary, we make the following contributions:

1. We propose a specific scoring function for content and profile diversification using a probabilistic model.
2. We propose a greedy threshold-based *top-k* algorithm to process queries using our profile diversity score using the concept of candidate list.
3. We propose various techniques for optimizing the computation of *top-k* diversified profiles.
4. To evaluate the benefits of our scoring function and optimization techniques, we ran our algorithms using three datasets: two from *Delicious* and one from *Flickr*. The results show that our approach increases the overall quality of recommendations and that our optimizing strategies reduce significantly the response time of the diversified *top-k* computation.

This paper is a major extension of [5], with more than 50% of new material, including new optimization techniques for *top-k* diversified profile computation, supported by new experiments. More precisely, in Section 5 we first propose a simplified profile diversification model to speed up the documents score computation. Then, we propose strategies to limit the number of diversification scores, which should be computed during query processing. Afterwards, we propose two techniques, in which the score of an item in the index takes into account the diversification requirements, thus reducing significantly the number of accesses to the inverted lists done by the *top-k* algorithm. We also propose two techniques exploiting users' feedbacks to tune the diversification level. Finally, in Section 6, we report intensive experiments showing the benefits of our techniques.

This paper is organized as follows. Section 2 provides some background on search and recommendation in the context of on-line communities and presents the problem definition. Section 3 describes our general scoring function, *ProfDiv*, based on probabilistic diversification. Next, Section 4 presents all algorithms necessary for *ProfDiv*. Section 5 presents a set of optimization techniques for *ProfDiv*. In Section 6, we compare *ProfDiv* with state-of-the-art diversification techniques and show the benefits of our optimization techniques. Section 7 presents the related work, and finally, Section 8 concludes and provides directions for future work.

2. Basic Concepts and Problem Definition

In this section, we introduce the background necessary to understand the problem we address.

In the context of recommendation retrieval, we choose a hybrid recommendation approach combining *content based* [6] and *collaborative filtering* [7] where a user's profile - or alternatively user's interests - is defined based on the content of the items $I_i = \{it_1, ..., it_m\}$ she shares. Thus, we assume a set of users $U = \{u_1, ..., u_n\}$. An item it can be shared by 1 to n users. The popularity of a given item is the number of users sharing it. Here, we exploit the idea of recommendation retrieval in the sense that users submit queries to search among recommended items shared by similar users.

Items are represented in a vector space model [8, 9]. By using $tf \times idf$ an item is represented by a list of keywords $k_1, ..., k_z$, and the vector represents the weight of each distinct keyword given the item and the whole corpus. A user $profile_i$ expresses his interests based on I_i. In fact a user's profile is the average of the $tf \times idf$ vectors of the items the user shares. Queries are expressed by a list of keywords $k_1, ..., k_z$.

Problem Statement: Given a set of users U, a set of items I and a keyword query q submitted by some user u, the problem we address is to efficiently recommend to the user u the *top-k* most relevant and diversified items $R^q \in I$. We assume that the k items are sorted in decreasing order of relevance in the results list R^q.

The intuition behind our approach is that diversification guarantees can be enhanced by diversifying the items and related users profiles in R^q. Therefore to produce R^q we identify four search and recommendation requirements with respect to an item it_i:

1. The similarities of it_i and q.
2. Content diversification with respect to the items already chosen in R^q.
3. The popularity of it_i.
4. Profile diversification with respect to the profiles of the users that own the items already chosen in R^q. Those profiles should be either similar to u or similar to q.

3. Scoring Model

Several methods have been proposed for diversification [10, 11, 12, 1, 13]. However, they only address requirement 2 discussed in the previous section. Our goal is to introduce profile diversification (*i.e.* requirement 4), taking into account a probabilistic diversification model because it provides more guarantees for diversification quality, as we show in our experiments in Section 6. In Section 3.1, we present the probabilistic model used for diversification, and in Section 3.2, we introduce our profile diversification approach.

3.1. Probabilistic Diversification

In the domain of information retrieval, given I and a query q, the computation of the *top-k* diversified items is known to be an *NP-hard* problem. Following [12, 11], $div(it_i|\{it_1, ..., it_{i-1}\})$ is defined as the diversification probability of it_i (*i.e.* brings novelty to the user u) with respect to the previously chosen items in R^q (i.e. $\{it_1, ..., it_{i-1}\}$). In this model, the diversity can be expressed using the notion of redundancy. The redundancy $red_c(it_i, it_j)$ is computed by comparing the similarity between it_i and it_j. Angel and Koudas [11] define the diversity probability as $1 - red_c(d_i|it_1, ..., it_{i-1})$. Based on the hypothesis that the redundancy between items it_i and it_j is independent of its redundancy with the other items [14, 11, 12], the probabilistic diversification score is defined as:

$$1 - red_c(it_i|it_1, ..., it_{i-1}) = \prod_{it_j \in \{it_1, ..., it_{i-1}\}} (1 - red_c(it_i, it_j))^{\alpha} \tag{1}$$

where $\alpha \in [0..3]$ is a parameter that enables to tune the diversification level, such that $\alpha = 0$ means no diversity, and higher values enable more diversity. The default value for this parameter is 1. In Section 4.3, we introduce two approaches to compute good values for α based on user feedbacks.

3.2. ProfDiv Scoring

To address the four requirements presented in Section 2, we propose a new diversified scoring function:

$$score_{ProfDiv}(it, u, q) = rel(it, q).div_c(it|\{it_1, ..., it_{i-1}\}).div_p(u_{it}|\{u_{it_1}, ..., u_{it_{i-1}}\}) \quad (2)$$

$rel(it, q)$ defines the probability that it will answer the query q. It can be defined as the similarity measure between it and q (*e.g.* using cosine, jaccard, etc.)[9]. This addresses requirement 1.

$div_c(it|\{it_1, ..., it_{i-1}\})$ is a straightforward application of equation 1 and addresses requirement 2.

$div_p(u_{it}|\{u_{it_1}, ..., u_{it_{i-1}}\})$ is the profile diversification score of item it and takes into account the item's popularity (requirement 3) and the diversification among trusted users (requirement 4). For each user in U holding a replica of it, is evaluated a trust and a diversification score (requirement 4) with respect to R^q.

The trust, noted $rel_{trust}(v_n, u, q)$, is a value which indicates the confidence the user u can have in the user v. Such information can be computed in many ways (*e.g.* social friendship, localization, previous recommendations, etc.). In this paper, we consider that the trust takes into account the relevance of the user v, given u and q. The relevance indicates if v is either similar to u (*i.e.* personalisation) or to q. (*i.e.* relevant to answer q). We define the user's relevance in Equation 3.

$$rel_{trust}(v, u, q) = \alpha \times sim(u, v) + (1 - \alpha) \times sim(v, q) \quad (3)$$

More formally, we propose the user profile diversification score defined in Equation 4. Recall that the profile diversification score also takes into account the popularity of the item it_i (requirement 3), this is why we need $\frac{1}{N}$. Notice that $\frac{1}{N}$ is also used for normalization.

$$div_p(u_{it}|\{u_{it_1}, ..., u_{it_{i-1}}\}) = \frac{1}{N} \cdot \sum_{v_n \in u_{it_i}} \left[rel_{trust}(v, u, q). \prod_{v_m \in \{u_{it_1}, ..., u_{it_{i-1}}\}} (1 - red_p(v_m|v_n))^{\beta} \right] \quad (4)$$

where $\beta \in [0..3]$ is a parameter that enables to tune the profile diversification level, such that $\beta = 0$ means that the results are not shared, and higher values of β mean that items should be shared by more diverse users. The default value for this parameter is 1. In Section 4.3, we discuss on choosing good values for β based on user feedbacks.

4. Computing Diverse Results

In this section we present in detail the algorithms involved in profile diversification. For the sake of
120 clarity, in section 4.1, we present the adapted version of the algorithm related to the probabilistic model we adopt [11]. In Section 4.2, we propose a new algorithm to compute profile diversification. Finally, in Section 4.3, we introduce two approaches to compute the diversification tuning parameters α and β.

4.1. top-k Algorithm

In [11], the authors propose an algorithm (called *DAS*) using the following scoring function:

$$rel(it_i, q).(1 - red(it_i|\{it_1, ..., it_{i-1}\})) \tag{5}$$

DAS is a threshold based algorithm. Given a query q and a set of items I, a threshold algorithm operates over a set of inverted indices:

$$\begin{aligned} &w_i \Rightarrow < it_a, sc_a >, < it_b, sc_b >, ..., < it_n, sc_n > \\ &... \\ &w_m \Rightarrow < it_e, sc_e >, ..., < it_n, sc_n > \end{aligned} \tag{6}$$

where w_i is a word, it_a an item and sc_a the score of the item with respect to the word w_1 (*i.e.* $sc_a = sim(w_1, it_a)$). The items are sorted in decreasing order of sc. The score of an item in an inverted list is called *indexing score*. Notice that the set of indices used by the threshold algorithm depends on the query q. For instance, if $q = \{w_i, w_m\}$ then the inverted indices will be the ones of w_i and w_m. Finally, the algorithm stops when the threshold condition δ is satisfied. δ is computed based on the inverted indices:

$$\delta = f(s_1, s_2, ..., s_n) \tag{7}$$

where f defines a specific measure (*e.g.* cosine, etc.) and s_i is the last sorted access on the w_i index. For instance, given a set of inverted indices $\{w_i, w_j\}$, suppose we want to retrieve the *top-1* item. The stop condition will be satisfied if the score of an item it is superior or equal to $\delta = f(s_i, s_j)$.

ProfDiv is derived from *DAS*. The main difference is the concept of candidate list. The goal is to find an optimal list R^q of k items such that we cannot find a better list R_2^q given u and q and our scoring function. That is, given R^q and an item $it_i \in R^q$, where $i \in \{1, ..., k\}$, we cannot find any item $it_j \notin \{it_1, ..., it_{i-1}, it_i\}$ that would have a better score than it_i at the i^{th} place in R^q.

The pseudo-code of *ProfDiv* is shown in Algorithm 1. The algorithm runs until R^q contains k items (line 2). From line 3 to 5, the algorithm performs a sorted access to get the next item, then it computes its score (*i.e.* $score_{ProfDiv}$, formula 2) and inserts it into a candidate list. The *candidate list* contains each item that has already been analyzed but that cannot be inserted in R^q, because the algorithm may still find items with better diversity score. Notice that an item's score is not fixed until it has been added to R^q. At line 6, *ProfDiv* checks if the best candidate has a score higher than the threshold δ. In other words, it checks if there isn't any better item in the indices. In that case, *ProfDiv* inserts the best item in R^q and updates the score of the other candidates (line 7 & 8).

4.2. Profile Diversification

In this section, we present how we compute div_p (Algorithm 1, line 4).

Algorithm 2 presents a method for computing div_p. From line 1 to 7, it computes for each user holding a replica of the item it a trust and a diversification score. On line 3, it evaluates the trust score of v_n with

Algorithm 1: *top-k ProfDiv*

```
Input: index, query, user, k
Output: the top-k most relevant items wrt. to our scoring function.
1 $R^q \Leftarrow$;
2 while $size(R^q) < min(k, size(corpus))$ do
3     $it \Leftarrow index.nextSortedAccess()$;
4     $it.score = rel(it, q).div_c(it|\{it_1, ..., it_{i-1}\}).\ div_p(u_{it}|\{u_{it_1}, ..., u_{it_{i-1}}\})$;
5     add it to candidates;
6     if the best candidate's score is higher than $\delta$ then
7         add best candidate to $R^q$;
8         Update the score of the other candidates;
```

respect to u and to q. Then, from line 4 to 6 it evaluates the diversification score of v_n with respect to the users that hold an item already inserted in R^q.

Finally, in line 7, it combines the trust and the diversification score and adds the computed value to the global profile diversification score. Line 8 normalizes the value of div_p and takes into account the popularity of it_i.

The number of iterations is equal to:

$$|U_{it_i}|.|U_{[it_1,...,it_{i-1}]}|$$

where $|U_{it_i}|$ refers to the number of users sharing the item it_i. Thus, the complexity of the function, in the worst case, is $O(n^2)$, where n is equal to the total number of users.

Recall that the profile redundancy score between two items takes into account the trust score which depends on the user u submitting the query. Therefore the profile diversification cannot be precomputed because a specific index would be necessary for each user [15, 16].

4.3. Considering User Feedbacks

In Section 3.1, we introduced two diversification tuning parameters: α and β. While the default value for these parameters is 1, it is not obvious which value will maximize the user satisfaction. In this section, we propose the idea of exploiting users' feedbacks to obtain better results.

Each time a query is submitted, users can provide feedbacks about the diversification level. The feedbacks are done on the query results in the form of "more diversity or less diversity is needed", until achieving the desired diversity. Our goal is to aggregate the feedbacks to define good values for α and β parameters. For this, we propose the following approaches:

1. **Per-user diversification tuning:** each time a user gives a feedback about a query, the values of α and β, corresponding to the preferred level of diversification are saved to the user profile. Then, during

Algorithm 2: Profile Diversification Score Computation

```
Input: $List[it_1, ..., it_{i-1}]$,User $u$,Query $q$,Item $it_i$
Output: The profile diversity score of $it_i$ wrt. $q$, $u$ and $[it_1, ..., it_{i-1}]$
/* the items are indexed based on sim(d,q).                                   */
1 $profDiv \Leftarrow 0$;
2 for $v_n$ in $U_{it_i}$ do
3     $t \Leftarrow trust(u, v_n, q)$;
4     $div \Leftarrow 1$;
5     for $v_m$ in $U_{[it_1,...,it_{i-1}]}$ do
6         $div \Leftarrow div \times red(v_n, v_m)$;
7     $profDiv \Leftarrow profDiv + t \times div$;
8 $profDiv \Leftarrow \frac{profDiv}{N}$;
```

query processing, the values of α and β are set to a value computed from the user's profile, e.g. the average or the median of the previous feedbacks. With this approach, each user has a different level of diversification.

2. **Per-query diversification tuning:** each time a user gives a feedback about a query $q = k_1, ..., k_p$, the values of α and β are saved and aggregated to the previous values of the inverted lists corresponding to the words $k_1, ..., k_p$. The aggregation method can be average or median. Then, during the query processing, whenever a user submits a new query $q' = k'_1, ..., k'_p$, the values of α and β are computed as the aggregation of their values in the inverted lists corresponding to $k'_1, ..., k'_p$. With this approach, all users have the same diversification level, but different queries may have different diversification levels.

5. Optimizations

In this section, we propose a set of optimizing techniques to improve the performance of the approach proposed in the previous section. First, in Section 5.1, we simplify the scoring model to reduce its computational complexity. Then, in Section 5.2, we propose a threshold for the diversified *top-k* algorithm, to reduce the number of accesses in the inverted lists. In Section 5.3, we propose two techniques to reduce the number of elements in the candidate list and therefore the number of diversified scores to compute. In Section 5.4, we propose two different indexing scores (*i.e.* the score used to sort the items in the inverted lists) that take into account some diversification requirements. Finally, in Section 5.5, we propose an adaptive indexing approach to reduce the number of accesses in the index dynamically based on the queries workload.

5.1. Simplified Profile Diversification Score

Computing the profile diversification score, based on Equation 4, has a complexity of $O(n^2)$, where n is the total number of users, as shown in Section 4.2. This complexity may be high, particularly for online applications. In addition, since the profile diversification is personalized (*i.e.* trust), it cannot be precomputed as it would require an index per user [16, 15]. This is why we propose a simplified approach for computing the score.

The intuition behind our simplification approach is the following: during query processing, when accessing a new item, instead of computing a profile diversification and a trust score per user and aggregating the results at the end, we propose to aggregate the user's profile during the indexing step first and to compute the profile diversification afterwards. Thus, during query processing, the profile diversification score can easily be computed without iterating over all users sharing the items.

In this approach, each item is associated to a bi-dimensional vector where the first dimension represents the content of the item (*i.e.* the $tf \times idf$ vector of the item) and the second dimension represents the aggregated profiles of the users sharing the item. Let T_u be the $tf \times idf$ vector of the tags submitted by user u (*i.e.* the vectorial profile of user u) and U_{it} the set of users sharing item it. The aggregation function f used in our approach is the average of tags:

$$f(it) = \frac{\sum_{u \in U_{it}} T_u}{|U_{it}|} \tag{8}$$

Since $f(it)$ returns the same vector no matter who is the user submitting the query q, it can be precomputed. Based on this item's representation, we propose a simplified profile diversification score with a complexity of $O(k)$:

$$div_p(it_i|\{it_1, ..., it_{i-1}\}) = rel(u, f(u_{it_i}), q) \times \prod_{it_j \in \{it_1, ..., it_{i-1}\}} 1 - red(f(u_{it_i})|f(u_{it_j})) \tag{9}$$

In the above equation, $rel(u, f(u_{it_i}), q)$ is the relevance of the users sharing it_i taking into account both the user u and its query q and $red(f(u_{it_i})|f(u_{it_j}))$ is the redundancy of the users sharing it_i with respect to the users sharing it_j. Thus, the score of an item can be formulated within a generic diversified framework:

$$score(it_i, u, q) = rel(it, u, q) \times div_{c/p}(it_i|\{it_1, ..., it_{i-1}\}) \tag{10}$$

where $rel(it, u, q)$ takes into account the relevance of item it_i and of the users sharing the item and $div_{c/p}$ is the diversification of both the items content and the profiles of the user sharing the items in the results list.

5.2. Refined Threshold

195 One of the main limitation of Algorithm 1 is the number of accesses in the inverted lists needed to compute the results list. As presented in formula 7, the threshold δ is evaluated using the item's score in the indices $\{w_1, ..., w_n\}$. In Algorithm 1, div_c and div_p (and $div_{c/p}$) are always smaller than 1. Also, notice that while the number of items in the results list R^q grows, the content diversification score and the profile diversification score become smaller for any given item $it_i \notin R^q$. For instance, in our experiments, even if all items, which will be added to R^q, have already been accessed and added to the candidate list, the algorithm would still perform a high number of additional accesses in the list.

We propose to use a new threshold δ' with respect to our scoring function to optimize the number of sorted accesses:

$$\delta' = f(s_1, s_2, ..., s_n).f_{div_c}(it_i, \{s_1, s_2, ..., s_n\}).f_{div_p}(it_i, \{s_1, s_2, ..., s_n\}) \tag{11}$$

Notice that to compute f_{div_c} and f_{div_p}, we need additional information because the indices $\{s_1, ..., s_n\}$ are not sufficient. For this, we use four functions whose outputs can be precomputed:

1. max_div_c: returns the maximum content diversity score between it_i and the items that follow it_i in $\{s_1, s_2, ..., s_n\}$.
2. max_div_p: returns the maximum profile diversity score between it_i and the items that follow it_i in $\{s_1, s_2, ..., s_n\}$.
3. max_trust: returns the maximum trust score of the users that share the item in $\{s_1, s_2, ..., s_n\}$. Notice that the part of the trust score, which depends on the user u that submitted the query, is equal to 1.
4. max_rep: returns the maximum number of replicas of items involved in $\{s_1, s_2, ..., s_n\}$.

Using the above functions, we compute f_{div_c} and f_{div_p} as follows:

$$f_{div_c}(it_i, \{s_1, s_2, ..., s_n\}) = \prod_{it_j \in \{it_i, ..., it_i - 1\}} max\_div_c(it_j) \tag{12}$$

$$f_{div_p}(it_i, \{s_1, s_2, ..., s_n\}) = \frac{max\_rep}{N}.max\_trust. \prod_{it_j \in \{it_i, ..., it_i - 1\}} max\_div_p(it_j) \tag{13}$$

Lemma 5.1. *The content diversity score of a given item it_i is less than or equal to f_{div_c}.*

Lemma 5.2. *The profile diversity score of a given item it_i is less than or equal to f_{div_p}.*

The proof of these lemmas can be done by a simple inequality based on our scoring model where each subpart of the function is replaced by its maximum possible value.

Notice that $\prod_{it_j \in \{it_i, ..., it_i - 1\}} max\_div_c(it_j)$ and $\prod_{it_j \in \{it_i, ..., it_i - 1\}} max\_div_p(it_j)$ can be calculated incrementally each time an item is added to the results list R^q.

We now present an example to compare δ with our new threshold. Due to lack of space and for simplicity, we simplify the scoring function of Algorithm 1 by removing the trust and the popularity related to div_p:

$$div_p(it_i) = \sum_{v_n \in u_{it_i}} \left[\prod_{v_m \in \{u_{it_1}, ..., u_{it_{i-1}}\}} (1 - red_p(v_m | v_n) \right] \tag{14}$$

Not surprisingly, removing $\frac{1}{N}$ and rel_{trust} from the scoring function, gives a threshold, δ'', which is quite simpler to compute compared to δ', but has the same general behavior:

$$\delta'' = lastSA. \prod_{it \in R^q} max\_div_c(it). \prod_{it \in R^q} max\_div_p(it) \tag{15}$$

Table 2 shows a running case in which Algorithm 1 builds R^q using δ''. We show that the number of sorted accesses would have been largely higher if we've used δ. The input is a built index of items based on a query.

Table 2: Example of the effect of the threshold on the number of sorted accesses.

Step	Sorted Access	$rel(it, q)$	Max div_c	Max div_p	Final Score	δ	δ'	R^q	C
1	ItemA	0.90	0.85	0.45	0.9	0.9	0.345	ItemA	-
2	ItemB	0.88	0.84	0.46	0.238	0.88	0.34	ItemA	ItemB
3	ItemC	0.87	0.95	0.65	0.34	0.87	0.33	ItemA,ItemC	ItemB
...									
n	ItemZ	0.55			0.0023	0.55			

The first column *step* corresponds to a whole iteration in Algorithm 1 (line 3 to 9). The second column *sorted accessed* indicates the sorted access done at the given step (line 3 of Algorithm 1) on the index of the input. The columns *max* div_c and *max* div_p indicate that the item's, on which we have just done the sorted access (*e.g.* item A for step 1), cannot be more diverse than the indicated value, with respect to all other indexed items not yet accessed. R^q is the list of results and C the list of candidates. The columns δ and δ'' indicate the value of the thresholds at the given step.

On step 1, Algorithm 1 performs a sorted access on $ItemA$. As it is the first item, the diversification score is 1 and the final score of the item is $rel(it, q) = 0.9$.
On step 2, Algorithm 1 performs a sorted access on $ItemB$. The final score of $ItemB$ is 0.238 due to its diversification score with respect to $ItemA$. Notice that δ'' (which is lower to δ) has a value of 0.34, which is higher than $ItemB$'s score. It means that we may find a better item.
Then, on step 3, Algorithm 1 performs a sorted access on item $ItemC$. The final score of this item (with respect to $ItemA$) is 0.34, which is higher or equal to δ''. Thus, there will not be any better item in the index. Therefore, $ItemC$ is inserted in R^q. Notice that δ is equal to 0.87, and Algorithm 1 couldn't insert $ItemC$ in R^q at this step by using δ. Furthermore, we can see that at step n, δ is equal to 0.55, which is still higher than $ItemC$'s score, thus with this threshold the algorithm should read all items of the lists. However, with our threshold, the algorithm stops after doing 3 sorted accesses. This confirms the fact that our new threshold approach provides important performance improvement.

5.3. Limiting the Candidate List Size

Computing the diversification score of the items in the candidate list (Algorithm 1, line 8) represents a huge proportion of the processing time. In this section, we propose two different strategies for limiting the candidate list size. First in Section 5.3.1, we show how to limit the size of the list to a maximum size, and, in Section 5.3.2, we propose an approach to limit the size of the list by filtering the items with respect to a dynamic threshold.

5.3.1. *Maximum Size Candidate List*

To reduce the number of diversification scores to compute, we can just limit the candidate list to a maximum size, *e.g.* N. To do this, we keep the candidate list C sorted in decreasing order of scores. Then, each time its size exceeds N, the algorithm removes the last item (*i.e.* the item whose score is the minimum) from the list.

To do so, we should change two functions in Algorithm 1. First, the `add` method, after inserting a new item to the list C, the algorithm should reorder C in descending order of scores, and remove the last item if the list size exceeds N. The `update` method should also reorder the list according to the scores, after each score update.

Although limiting the number of candidates incurs an additional overhead to the management of the candidate list, as it will be shown in Section 1, it allows us to drastically reduce the number of items whose diversification score should be computed, and therefore to reduce significantly the response time of Algorithm 1.

5.3.2. *Filtering Candidate List*

In this subsection, we propose a method to filter the candidates with respect to a threshold. Defining a static threshold does not make sense, since the diversified scores of the items in the candidate list are getting lower and lower when the results list size increases. In our approach, we choose a threshold that is a ratio of the best score in the candidate list. For instance, with a ratio of 0.1, all items whose score is less than 0.1 of the best candidate score until now will not be kept in the list.

To implement this approach, we modify both the `add` and `update` methods as in Algorithms 3 and 4 respectively.

Algorithm 3: Add an *item* to Candidate List

```
Input: $it$, $C$, $ratio$, $best$
Output: $it$ is added to the candidate list $C$ if its score is higher to $ratio \times score(best)$, and $best$ is
        updated if needed
1 if $score(it) > ratio \times score(best)$ then
2     add $it$ to $C$;
3     if $score(it) > score(best)$ then
4         $best = it$;
```

As shown in Section 6, our filtering approach reduces the number of diversified scores to compute while preserving a good quality for the algorithm's results.

Algorithm 4: Update Diversification Scores of Candidate List

```
Input: $C$, $it$, $ratio$, $best$
Output: $C$'s items have their scores updated wrt. $it$, and $C$ keeps only the items whose score is less
        than $ratio \times score(best)$
1 Update the score of $\forall i \in C$ wrt. $it$ and compute best;
  /* recall that the best candidate has just been added in the results list... $best$ is
     no longer up to date                                                              */
2 Remove $\forall items$ which score is inferior to $ratio \times score(best)$;
```

5.4. Diversified Indexing

Algorithm 1 is greedy, thus it may not return the optimal results list. Moreover, if the first item added to the results list is redundant with respect to all other items, then the results list quality will be decreased and the number of accesses in the index increased. Indeed, if the quality of the results list is reduced, the score of the last item will be reduced and the algorithm will need to perform additional accesses in the list, so that the threshold value becomes less than the last seen item's score. In this section, we propose a method that decreases the indexing score (*i.e.* the score of an item in a specific inverted list) of the very redundant items by giving them a low weight, in order to decrease their chance to enter to the results list.

In our approach, the weight is computed as follows:

$$weight(item, items) = \frac{\sum_{it \in items} sim(it, item)}{|items|} \tag{16}$$

In other words, the weight of an *item* is computed with respect to the set of *items*, which can be either the whole set of items in the system or be the set of items in the inverted list. Thus, the indexing score of the item in an inverted list is computed as follows:

$$indexScore(item, L_k/items) = rel(item, k) \times \frac{\sum_{it \in L_k/items} sim(it, item)}{|L_k/items|}^{a} \tag{17}$$

where a is a parameter allowing to control the importance of the weight. If an item is very relevant but also completely redundant with respect to all other items, its score will be decreased. The main drawback of this approach is that some items will be definitely ignored.

5.5. Adaptive Diversified Indexing

In this section, we propose a technique that adapts the indexing score of the items with respect to the system's workload, *i.e.* past queries.

Given an inverted list and a query q, we identify three categories of items:

- The relevant and diversified items that are both accessed in the index and returned to the user;
- The redundant items that are accessed in the index but not returned to the user;

- The items that are neither accessed in the index nor returned to the user.

The intuition behind our approach is that, given an inverted list and the query workload, some items will 285 always be in the second category (*i.e.* neither relevant nor diversified but accessed). These items will incur some overhead to the results list R^q computations, but will never improve the results list quality because they don't enter the result list.

Our idea is to index all items by taking into account their relevance with respect to the inverted list's keyword and a weight that depends on the frequency at which an item is redundant in the inverted list.

The approach is adaptive in the sense that it depends on the workload. The indices are dynamically updated each time a query is processed as presented in Figure 1. Given an inverted list L_k, an item $it \in L_k$

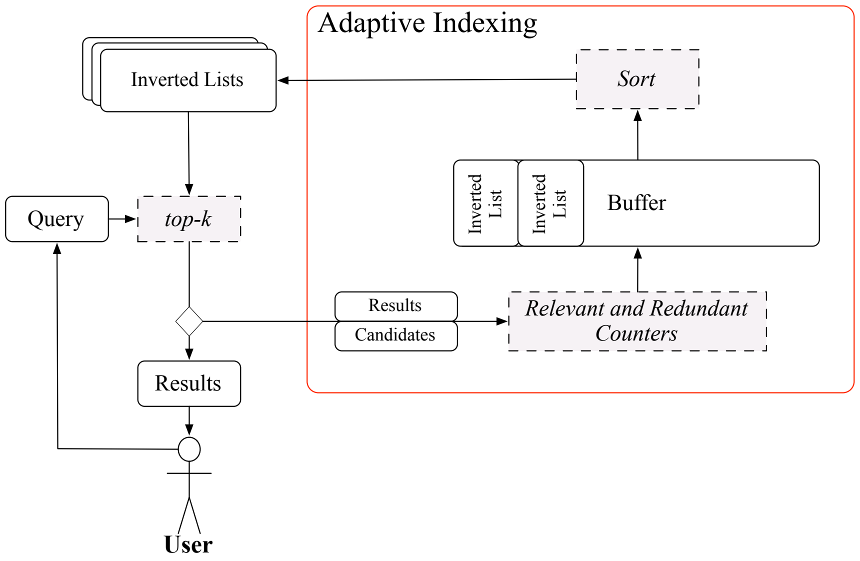


Figure 1: Adaptive indexing architecture.

is associated to both a relevance counter to know how many times it has been relevant and a redundancy counter to know how many times the item has been redundant.

At the end of query processing (*cf.* Algorithm 1), two lists of items exist: the results list R^q and the candidate list C^q. All items $it \in R^q$ belong to the first category of results and see their relevance score increased while all items $it \in C^q$ belong to the second category and see their redundancy score increased. Notice that an item cannot belong to the two lists at the same time. Based on these counters, the frequency at which an item is considered redundant can be computed as follows:

$$f_{red,L_k}(it) = \frac{|it_{redundant}|}{|it_{redundant}| + |it_{relevant}|} \tag{18}$$

Where $|it_{redundant}|$ is initialized to 0 and $|it_{relevant}|$ is initialized to 1. In each inverted list, the items are sorted with respect to the following score:

$$indexScore(item, L_k) = rel(item, L_k) \times W(f_{red,L_k}(item)) \tag{19}$$

Where $W(x)$ is a continuous decreasing function which has the following properties:

$$\begin{aligned} &W(0) \simeq 1 \\ &\lim_{x \to 1} W(x) \to w_{min} \end{aligned} \tag{20}$$

Where $0 < w_{min} < 1$. In other words, the more an item is redundant the lower its score in the inverted list will be. Therefore, if an item is always redundant, it will be pushed at the end of the inverted list and will be less and less accessed, thus leading to better response time during query processing. Notice that the weigh function has a system defined minimum value w_{min} strictly higher than 0 which means that an item cannot be removed from the index.

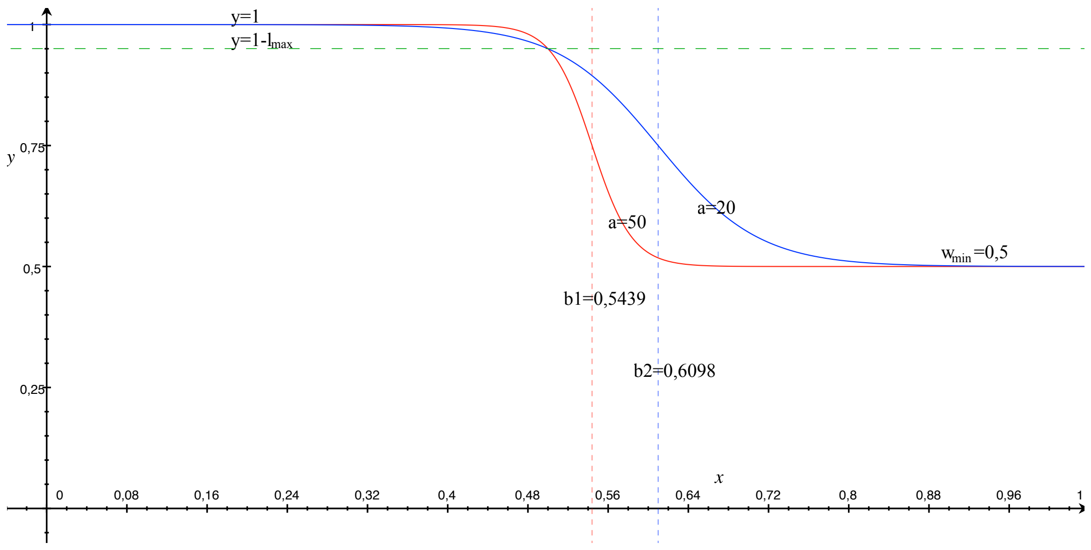


Figure 2: Weight function $W(x)$ with respect to the various parameters.

In our approach, $W(x)$ is a sigmoid function. A sigmoid function [17] is characterized by two stable states (asymptotes) at the beginning and at the end (in our case $y = 1$ and $y = w_{min}$) and a transitional state that joins the two stable states as presented in Figure 2. The weight function is defined as follows:

$$W(x) = \frac{1 - w_{min}}{1 + e^{a \times (x-b)}} + w_{min} \tag{21}$$

where w_{min} is the worst weight an item can receive, b enables to shift the transitional states, and a allows to modify the duration of the transitional state. The variable x is the redundancy frequency of a given item as presented above.

The user has to specify the maximum redundancy frequency f_{max} ($0 < f_{max} < 1$) such that all items whose redundancy frequency ranges from 0 to f_{max} have the guarantee that their worst loss in terms of indexing score will be less than l_{max} knowing that $l_{max} \to 0$ and $l_{max} > 0$. In other words, items which are periodically relevant will have weight score higher than $1 - l_{max}$ and therefore $indexScore(item, L_k) = rel(item, L_k) \times W(f_{red,L_k}(item)) \simeq rel(item, L_k)$. However, if an item is too redundant (or $x > f_{max}$), its weight will start decreasing and the same for its indexing score. Based on these variables, the parameter a follows the following inequality:

$$a \geq \frac{2 \times ln(\frac{l_{max}}{1-l_{max}-w_{min}})}{f_{max}-1} \tag{22}$$

Small values of a result in large transitional states in the weigh function. Big values of a result in small transitional states. The parameter b can be expressed with respect to a:

$$b = f_{max} - \frac{ln(\frac{l_{max}}{1-l_{max}-w_{min}})}{a} \tag{23}$$

Because all items have their score decreased with respect to $W(x)$ – including the relevant ones –, the threshold might under-estimate the score of some items since their real score is higher and the threshold condition may be satisfied before accessing all relevant items. This is desired for redundant items, but not for relevant and diversified items. Based on the user defined parameter f_{max}, we know that all items whose redundancy frequency is less than f_{max} would not be missed. Additionally, we know that $W(f_{max}) = 1-l_{max}$. This enables us to define the following threshold:

$$f'(s_1, ..., s_n) = \frac{f(s_1, ..., s_n)}{1-l_{max}} \tag{24}$$

where s_i is the last sorted access on the list i.

Based on the definition of the threshold and of the weighed function, we propose the two following guarantees in terms of performance and results quality with respect to $W(x)$:

Theorem 5.1. *Given an inverted list L_k, if a query q represents $h\%$ of the queries that access L_k, and if $h \geq 1 - f_{max}$, then the results of q are guaranteed to be identical to an algorithm using a static index.*

Theorem 5.2. *Given l_{max}, a query q, $nbSA_a$ and $nbSA_b$ the number of sorted accesses that will be done to process q on respectively an adaptive index or a static index. If q represents at least $h\%$ of the queries that access L_k, and if $h \geq 1 - f_{max}$, then the probability that the algorithm performs better using an adaptive index instead of a static index, tends to* 1 *when l_{max} approaches zero, as shown in Equation 25.*

$$\lim_{l_{max} \to 0} P(nbSA_a < nbSA_b) = 1 \tag{25}$$

Both theorems are proved in Appendix AppendixA.

6. Experimental Evaluation

In this section, we provide an experimental evaluation of our approaches to assess the quality of recommendations, content diversification, profile diversification and of the algorithms' efficiency. We have conducted a set of experiments using a dataset from *Delicious*, an enriched version of the *Delicious* dataset and a dataset from *Flickr*. In Section 6.1, we first describe the experimental setup. Then, in Section 6.2, we discuss the results.

6.1. *Experimental Setup*

315 Our experiments have been conducted on three different datasets. The first dataset consists in 30,000 bookmarks downloaded from *Delicious* associated to about 55,000 unique tags submitted by about 2,000 users. From this dataset, we have created a second one by downloading the `html` content of each bookmark from the web. The third dataset contains 3.25 million images from *Flickr* associated to 3.5 million set of tags (*i.e.* 1 to 15 in general) submitted by 272,000 users. In all experiments, the items are indexed using all available keywords (*i.e.* title, tags, description or html content).

Our experiments consist in submitting queries and measuring some parameters about the query execution and the results returned by the algorithms. The queries were either automatically built or submitted by the users during a *user survey*. In the first case, to build the queries, we have associated together every tag submitted by a single user on a single bookmark or image on a single day. Each query is associated to the user who has initially submitted the tag (and therefore is associated to that user's profile).

The code used in our experiments was developed in *C++* and executed on a *core-2-duo 2.5Ghz* machine with *8GB* of *DDR3 RAM*. It can be downloaded at the following address:

`http://www.lirmm.fr/~servajean/DiversitySearch.tar.gz`

6.1.1. *Quality Evaluation*

330 To evaluate the quality of recommendations and diversification, we have proceeded in two steps: an automatic *numerical evaluation* and a *user survey*.

The automatic numerical evaluation consists in submitting all queries on different *top-k* algorithms and to measure several metrics. The queries are submitted by the users associated to them. Notice that the results are filtered to not take into account the items shared by the query's initiator. In our experiments, we compared the following algorithms:

1. `Probabilistic content diversification`: the probabilistic content diversification described in Section 3.1.
2. `Max-Min`: the minimum content diversification score between any two items in the results list is maximized.
3. `Max-Sum`: the sum of the content diversification score of an item with respect to all other items is maximized.
4. `Profile diversification`: the full scoring function where the trust score is fixed to 1.
5. `Personalized profile diversification`: the full scoring function.

and we have measured the following metrics:

1. `Relevance`: the average similarity between any item in the results list and the query:

$$rel = \sum_{it \in R^q} \frac{rel(it)}{|R^q|}$$

2. `Content diversification`: the average dissimilarity between any item in the results list with respect to the other items in the results list:

$$div_c = \sum_{it_i \in R^q} \sum_{it_j \in R^q} \frac{1 - sim(it_i, it_j)}{|R^q|^2}$$

3. `Profile diversification`: the average dissimilarity between the users that share an item in the results list and all users sharing the other items of the results list:

$$div_p = \sum_{it_i \in R^q} \sum_{it_j \in R^q} \frac{1 - sim(f(u_{it_i}), f(u_{it_j}))}{|R^q|^2}$$

4. `Trust`: the average similarity between the user submitting the query and the users sharing the items in the results list.

$$trust = \sum_{it \in R^q} \frac{rel(u, f(u_{it_i}), q)}{|R^q|}$$

The user survey has been realized in three steps. The first step's goal was to evaluate the quality of profile diversification while the second step's goal was to evaluate the quality of the personalized profile diversification (*i.e.* profile diversification and trust). The survey has been realized both for the *Delicious* and the *Flickr* datasets. Since the first part of the survey focused mainly on the enhancement of the diversification quality due to the profile diversification, we compared the following algorithms:

1. `Simple-topk`: we only take into account the similarities between the query q and the item.
2. `Content diversification`: we select items that are both relevant with respect to the query and diversified with respect to the other items in the results list taking into account the tags of the items.
3. `Profile diversification`: the full scoring function where the trust score is fixed to 1.

During the second part of the user survey, we have compared the following algorithms:

1. `Personalized content diversification`: the probabilistic content diversification score associated to a trust score.
2. `Profile diversification with no trust`: our scoring function where the trust score is fixed to 1.
3. `Personalized profile diversification`: our scoring function.

Finally, in the third part of the survey, we tried to learn from user feedbacks to tune the diversity per user, or per query, as presented in Section 4.3. To analyze the effect of tuning, we compared the following methods :

1. `Simple-topk`: we only take into account the similarities between the query q and the item.
2. `Content diversification`: we select items that are both relevant with respect to the query and diversified with respect to the other items in the results list taking into account the tags of the items.
3. `Profile diversification`: the full scoring function where the trust score is fixed to 1 and where **tuning parameters for diversity** are updated based on the feedbacks.

We have asked the users the following questions (the answer to the questions 1-2 was a number between 1 to 5):

1. How relevant is the results list with respect to the query?
2. How diversified is the results list with respect to the query?
3. Finally, the user was asked to rank the three results list from the best one to the worst one in terms of relevance and diversity. Two lists could be similar in terms of quality.

To evaluate a trust score, each user had to have a profile. To build their profile, the users were asked to submit a few queries and to select items they liked. Once these few queries were submitted, the evaluation could start. Notice that the users were never aware of which algorithm generated which list. Additionally, the lists were randomly presented to the users.

We have interrogated 20 users that had to evaluate more than 190 different queries. These queries were either suggested by the system or invented by the users.

6.1.2. Performance Evaluation

During the performance experiments, we have compared the following optimizations techniques:

1. `Refined threshold`: the threshold is refined taking into account additional information as presented in Section 5.2.
2. `Maximum size candidate list`: the size of the candidate list is limited to a maximum value as presented in Section 5.3.
3. `Filtered candidate list`: the candidates are filtered with respect to a dynamic threshold as presented in Section 5.3.
4. `Global diversified indexing`: the indexing score of each item depends on some global redundancy statistics of the item as presented in Section 5.4.
5. `Index diversified indexing`: the indexing score of each item depends on some inverted list redundancy statistics of the item as presented in Section 5.4.
6. `Adaptive Indexing`: the indexing score is dynamically updated with respect to the queries workload as presented in Section 5.5. This optimizations were run two times. One time with $f_{max} = 99\%$ and another time with $f_{max} = 99,9\%$.

and we have analyzed the following metrics:

1. Response time when the inverted lists are in memory.
2. Response time when the inverted lists are on disk.

To simulate the disk accesses, we have proceeded as follows. First, the indices are materialized in such a way that an access in an item in the index enables to load the entire vector's of the item. Thus, we avoid additional accesses to compute the diversification score between the items. We assume that every item's vector has a normalized length. We suppose that an item's memory size (the vector and its score with respect to a given inverted list) is $1KB$. Additionally, we suppose that a disk access requires a *seek* operation followed by a transfer operation. Each time the algorithm needs to access an inverted list on the

Operation	Characteristic
Seek Operation	duration: $9ms$
Data transfer	speed: $1Gb/s$
Number of blocs transferred by access	$64(\times 1KB)$

Table 3: Characteristics of an inverted access on the disk.

disk, it transfers several blocs at the same time to limit the number of seek operations. We have used the parameters presented in Table 3.

Notice that some optimization propositions have some configuration parameters (*e.g.* maximum size of the candidate list). In our experiments, these parameters are defined such that they avoid quality loss.

6.2. Experimental Results

6.2.1. Quality Results

The results of the first experiments are presented in Figure 3.

First, we can observe in Figures 3a and 3b that the *probabilistic model* and the two *profile diversification* methods achieve similar results in terms of relevance and content diversity while *Max-Min* and *Max-Sum* are about 10% worst in terms of diversification and slightly better in terms of relevance.

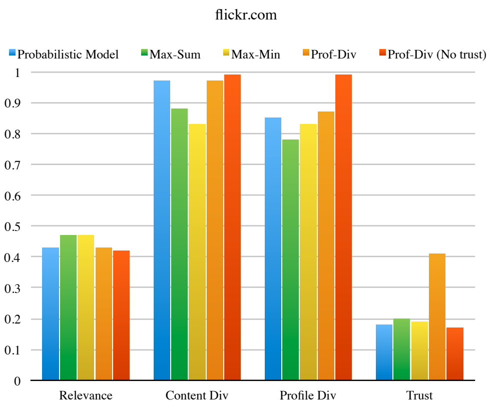
Additionally, Figure 3a (*i.e. Flickr* dataset) shows that our full scoring function enables a small increase of the profile diversification but also a huge increase of the trust with results more than two times superior compared to the other methods. The profile diversification achieves similar results in terms of trust but enables a gain of 15% in terms of profile diversification.

We can observe in Figure 3b (*i.e. Delicious* dataset) that our full scoring function enables a gain of 20% in terms of profile diversification compared to the other methods and a trust more than two times higher compared to the other methods. The profile diversification achieves a slightly worse trust score but enables to increase the profile diversification by more than 30%.

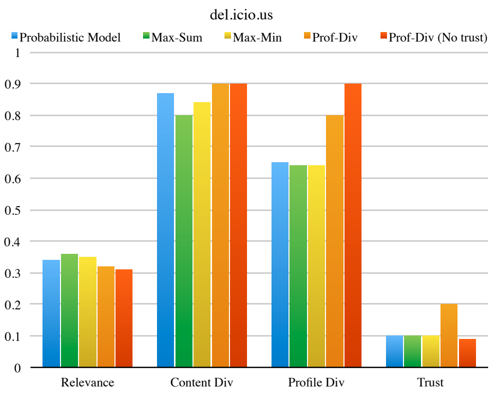
Although the numerical results of content diversification are similar for the content diversified algorithms and profile diversified algorithms, the user experience is different. Indeed, the tags are not always well defined and taking into account the user profiles (*i.e.* all the tags submitted by the user and not only those added to the current item) will enable a considerable gain in terms of user satisfaction. To evaluate this gain, we have realized two user surveys whose results are presented in Figure 4 and 5. In Figure 4, the analyzed method were: *un-diversified top-k*, *content diversified top-k* and *profile diversified top-k*.

Figures 4a and 4b show that the users find the list generated using profile diversification both more relevant and more diversified than the lists generated using the content diversification. Also, the users find the un-diversified list more relevant but less diversified than the two other methods.

In Figures 4c and 4d, the users were presented, several times, two results list and asked to choose the

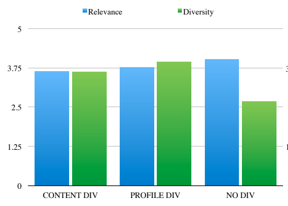


(a) Flickr dataset

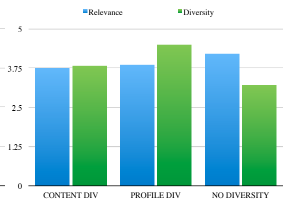


(b) Delicious dataset

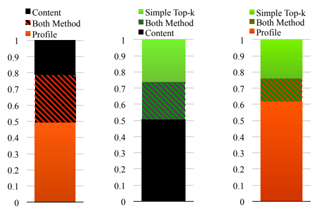
Figure 3: Automatic quality evaluation with the following metrics relevance, content and profile diversification and trust.



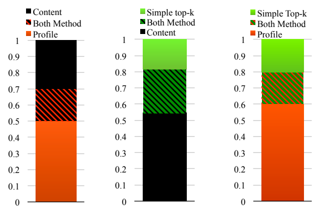
(a) Flickr dataset



(b) Delicious dataset



(c) Flickr dataset



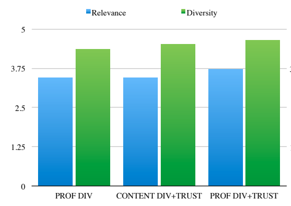
(d) Delicious dataset

Figure 4: Quality evaluation with the following scoring functions: relevance, content and profile diversification.

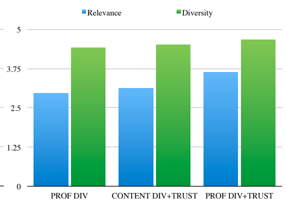
one they preferred. Each color represents a percentage of time a method was preferred to another while the striped color represents the percentage of time the user found both method similar.

Figures 4c and 4d show that in half of the cases, profile diversification was found better than content diversification. Also, in more than 60% of the cases, profile diversification was found better than un-diversified list. In 20% of the cases on the *Delicious* dataset and in 30% of the cases on the *Flickr* dataset, profile diversification was similar to content diversification. In 15 and 20% of the cases on the *Delicious* and on the *Flickr* datasets respectively, un-diversified lists and profile diversification were similar. Finally, in more than half of the cases, content diversification was better than un-diversified list, and in 20 and 30% of the cases on the *Delicious* and on the *Flickr* datasets respectively it was similar to un-diversified list.

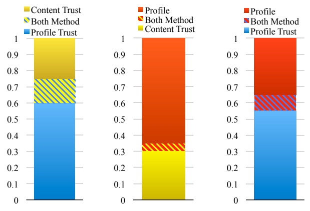
The analysis of the results leads to the following conclusion. In the majority of cases, diversity enhances the user satisfaction. Also, taking into account the users profile in diversification increases the quality of the results.



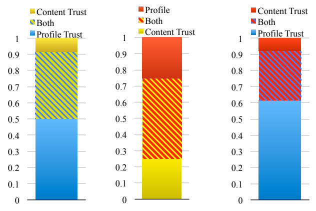
(a) Flickr dataset



(b) Delicious dataset



(c) Flickr dataset



(d) Delicious dataset

Figure 5: Quality evaluation with the following scoring functions: personalized content diversification and profile diversification with and without trust.

In Figure 5, the analyzed methods were: *profile diversity top-k*, *personalized content diversity top-k* and *personalized profile diversity top-k*.

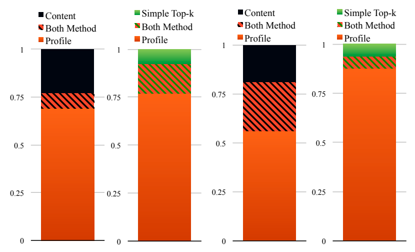
Figures 5a and 5b show that the users find all the methods quite similar in terms of relevance and redundancy. However, associating the trust and the profile diversification score performs the best score in both relevance and redundancy.

In Figures 5c and 5d, the users were presented, several times, two results list and had to choose the one they preferred. Each color represents a percentage of time a method was preferred to another while the striped color represents the percentage of time the user found both method similar.

Figures 5c and 5d show that in 50 and 60% of the cases on the *Delicious* and on the *Flickr* datasets respectively, personalized profile diversification was found better than personalized content diversification. Additionally, in 15 and 40% of the cases on the *Delicious* and on the *Flickr* datasets respectively, both methods were similar. In other words, in 75 and 90% of the cases on the *Delicious* and on the *Flickr* datasets respectively, personalized profile diversification is similar or better than personalized content diversification. The comparison between profile diversification and personalized content diversification returned very different results on the *Flickr* and *Delicious* dataset. Indeed, on the *Flickr* dataset, in 65% of the cases, profile diversification was preferred to personalized content diversification. On the *Delicious* dataset, both method were preferred in 25% of the cases and were found similar in 50% of the cases. Finally, in 55 to 60% of the cases, personalized profile diversification was considered better than profile diversification and in 10 and 35% of the cases on the *Delicious* and on the *Flickr* datasets respectively, both methods were similar.

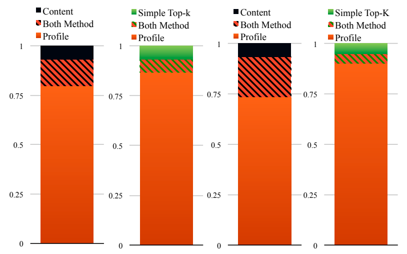
Two conclusions can be made. First, adding the trust score does not necessarily enhance the user satisfaction when using content diversification. Second, combining trust and profile diversification enables a huge gain in user satisfaction.

In the following, we investigate the effect of the diversity tuning parameters (*i.e.* α and β, *cf.* Section 4.3) on the results quality. The experiments are composed of a training set of queries and a testing set. To build the training sets, the users submitted queries and gave feedbacks about the diversification quality. Figures 6 and 7 show the results of our experiments. Figure 6 depicts the results when α and β are computed on



(a) Flickr dataset (b) Delicious dataset

Figure 6: Quality evaluation when the system adapts the diversity per users, using users feedbacks.



(a) Flickr dataset (b) Delicious dataset

Figure 7: Quality evaluation when the system adapts the diversity per query, using users feedbacks.

a per-user basis. Obviously, learning how much a specific user likes diversity enables a gain in terms of satisfaction, as it can be observed in the Figure. As Figure 6 shows, profile diversity is now preferred by the users in 77 to 82% of the cases. The gains are even higher when compared to a simple *top-k*.

Figure 7 presents the results when α and β are computed on a per-query basis. The gain obtained using this method is higher than that of the per-user basis. As shown, the users prefer profile diversity or find it similar to content diversity in 94% of the cases. This can be explained as follows. Different topics may use more or less synonyms. Some topics such as literature use a lot of synonyms and the results may seem to be diverse, while in reality they are not. Other topics, such as computer science, use less synonyms, and their content will be more redundant. Thus, learning from user feedbacks to know which query should provide more or less diversity enables a significant gain in terms of user satisfaction.

6.2.2. Performance Results

In this section, we present the results of our experiments done to evaluate the performance of the optimizations presented in Section 5. In these experiments, the trust score is fixed to 1. Figure 8 presents the results of these experiments when the scoring function takes into account profile diversification (simplified as presented in Section 5.1), and Figure 9 shows the results of these experiments when the scoring function takes into account content diversification.

Notice that each experiment is executed two times: one at small scale (*i.e.* 5,000 items indexed) and another at full scale (*i.e.* all items indexed).

Adaptive indexing approach has, on average, the best gain in terms of response time. Indeed, when taking into account profile diversification, Figure 8 shows that, at small scale, *adaptive* 99% and 99, 9% are about 33% faster on the *Flickr* dataset, and more than 2 times faster on the *Delicious* dataset when the inverted lists are in memory and from less than 3% of gains to more than twice as fast when the inverted lists are on disk.

At large scale, the gains are even bigger. *Adaptive* 99% and 99, 9% are more than 4 times faster when the inverted lists are in memory and are 1.47 to 3 times faster when the inverted lists are on disk.

By taking into account content diversification, Figure 9 shows that *adaptive* 99% *and* 99, 9% are 15% to more than 12 times faster on the *Flickr* and *Delicious* datasets respectively when the inverted lists are in memory. They are 15% to more than two times faster when the inverted lists are on disk on the *Flickr* and *Delicious* datasets respectively.

At large scale, the performance gains are still bigger. *Adaptive* 99% and 99, 9% are at least 2 times faster, and the gain can be up to more than 5 times.

About the loss provoked by the adaptive 99 and 99.9, both methods have similar levels of loss with respectively 0.704% and 0.644% at small scale and 1.02% and 0.82% at large scale. In other words, for both methods, most of the losts are due to queries whose results are returned by less than a query out of 1, 000. Notice that an adapted index could be combined to a normal index. Then, on query processing the system would choose, based on some query statistics (*e.g.* frequency), the right index to avoid any loss.

About the diversified indexing techniques (*i.e.* global or local to an index), both methods show the worst results in terms of performance. Indeed they do not exceed 2% of gain in terms of response time and have an average gain of 0%. More over, the *local diversified indexing* has a quality loss of more than 30% in average.

The gains of the refined threshold really depends on the dataset. For instance, the gains are close to 0 on the *Flickr* dataset while they are higher than 20% when used on the delicious dataset. Additionally, since this technique only refines the threshold, we have the guarantee that it will not result to a quality loss.

Finally, limiting the size of the candidate's list by choosing a maximum value or by filtering enables a performance gain from a few percentage to more than two times. However, since it only reduces the number of diversified score computation and not the number of inverted list accesses, when the lists are on disk, the gain is increased and goes from about 2% to more than 40%.

7. Related Work

Content diversity has been studied in Web search, database queries, and recommendations. Diversifying Web search results and recommendations aims to achieve a compromise between relevance and result heterogeneity. In [14], the authors adopt an axiomatic approach for diversity whose objective is to address

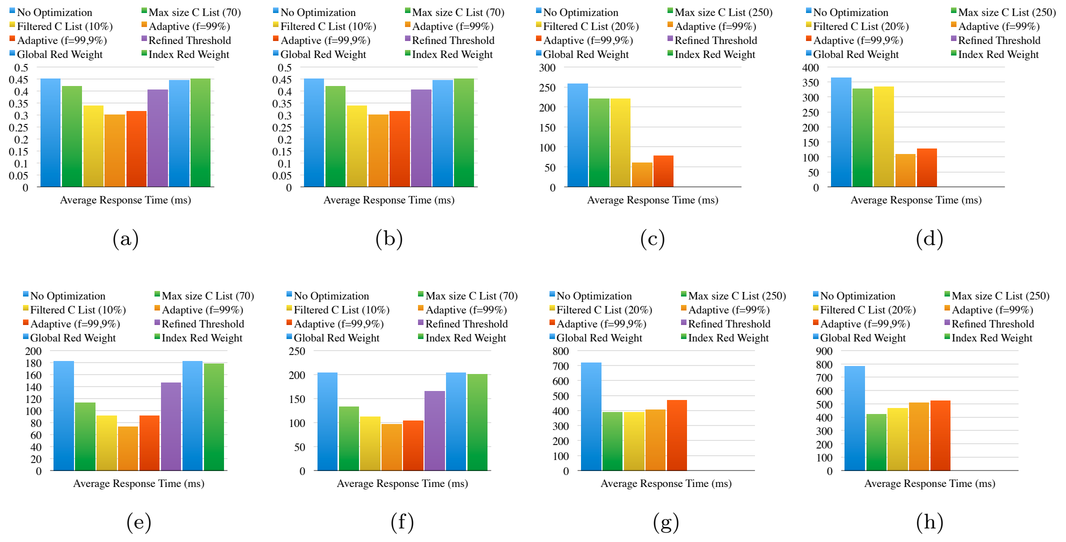


Figure 8: Performance evaluation of various optimizations using profile diversification: (a) Flickr dataset, 5,000 items in memory, (b) Flickr dataset, 5,000 items on disk, (c) Flickr dataset, 3,250,000 items in memory, (d) Flickr dataset, 3,250,000 items on disk, (e) Delicious dataset, 5,000 items in memory, (f) Delicious dataset, 5,000 items on disk, (g) Delicious dataset, 3,250,000 items in memory, (h) Delicious dataset, 3,250,000 items on disk

user intent. They show that no diversification function can satisfy all axioms together. In [18], taxonomies are used to sample search results in order to reduce homogeneity. In the database context [19, 20], some solutions propose to post-process structured query results, organizing them in a decision tree [20] for easier navigation. In [21], a hierarchical notion of diversity in databases is introduced, and efficient top-k processing algorithms are developed.

In some works on recommendation, *e.g.* [3, 22, 1], results are typically post-processed using pair-wise item similarity in order to generate a list that achieves a balance between accuracy and diversity. For example, the approach in [3] defines an intra-list similarity which relies the mapping items to taxonomies to determine topics or using item features such as author and genre. The method is based on an exhaustive post-processing algorithm which operates on a *top-n* list to compute the *top-k* results (n > k). In contrast, in [22], diversity is formulated as a set-coverage problem. [23] introduces diversity in the framework of sponsored search ads, proposing algorithms for the selection of ads intending to increase heterogeneity while not significantly reducing revenue and maintaining an incentive for advertisers to keep their bids as high as possible. Here, heterogeneity means that a single query can, in fact, be represented by several occurrences (*i.e.* meaning) and not only one. In [4], the authors propose a diversification technique based on the comments left by the users instead of the content of the items themselves. They propose diversification as a way to enhance news navigation. The diversity computation is done through hashing with *dLSH**. *DisC* [24] aims at returning a

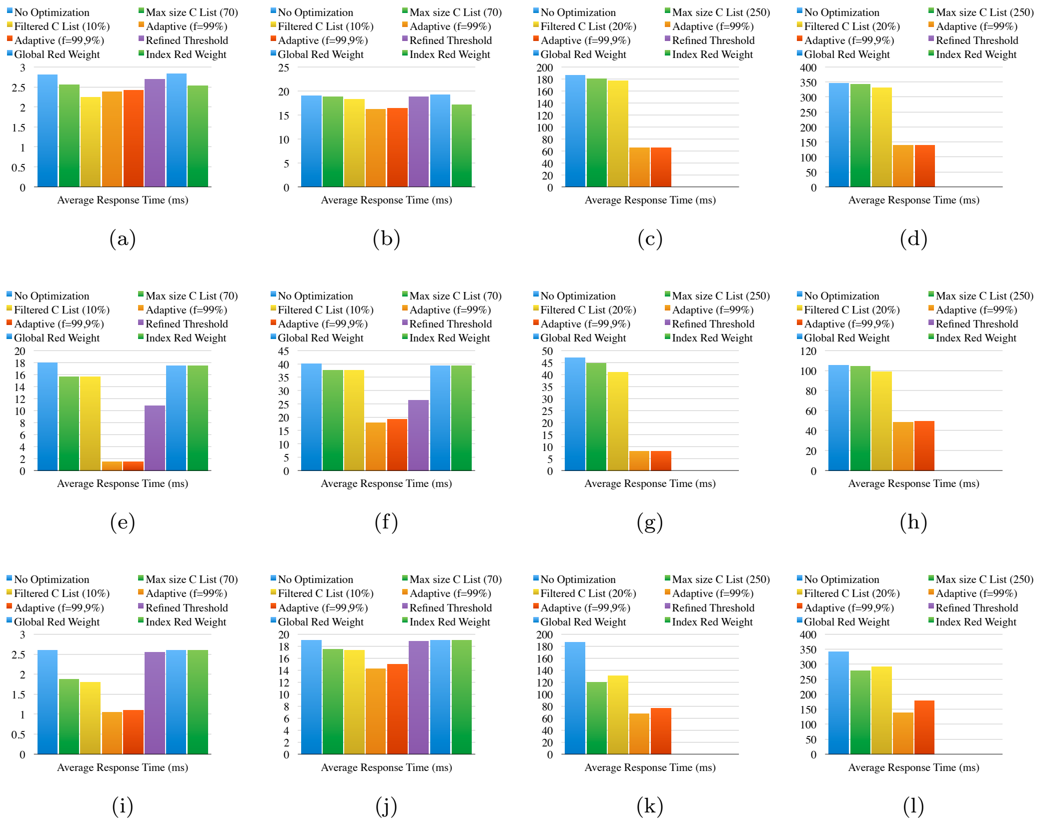


Figure 9: Performance evaluation of various optimizations using content diversification: (a) Flickr dataset, 5,000 items in memory, (b) Flickr dataset, 5,000 items on disk, (c) Flickr dataset, 3,250,000 items in memory, (d) Flickr dataset, 3,250,000 items on disk, (e) Delicious dataset, 5,000 items in memory, (f) Delicious dataset, 5,000 items on disk, (g) Delicious dataset, 3,250,000 items in memory, (h) Delicious dataset, 3,250,000 items on disk, (i) html Delicious dataset, 5,000 items in memory, (j) html Delicious dataset, 5,000 items on disk, (k) html Delicious dataset, 3,250,000 items in memory, (l)

set of items R such that every relevant item is represented by an item in R. In [24], diversity is not computed as a set of k items where k is user defined, but as a set of items such that the similarity of all relevant items with any item in R is less than a user defined value r (*i.e.* radius).

None of the above contributions tackles the problem of diversity as we do. We proposed to associate each item with the profiles of the users sharing it in order to compute the diversification score in a way that enhances the user satisfaction. We proposed some algorithms to efficiently compute the *top-k* best items taking into account, for instance, the system workload and *e.g.* past queries.

8. Conclusion

In this paper, we introduced profile diversity to enhance the quality of diversification in search and recommendation.

We proposed a scoring function that accounts for query relevance, content diversity to alleviate item similarity in query results, item popularity to account for community endorsements, and finally, profile diversity to expose users to items owned and shared by different types of users who have very high probability to share very different kinds of content.

Profile diversity provides more guarantees in terms of diversification quality. We support the profile diversification by recommending items that are shared by trusted and diversified users among all users. We proposed a new threshold algorithm suited for profile diversification. Our scoring function is based on a probabilistic model since it provides good guarantees of diversification. Additionally, we introduced the idea of considering users' feedbacks to tune the diversification level.

Additionally, we proposed a set of efficient optimization techniques to improve the performance of our solution. Our optimizations have focused on three issues : 1) simplification of the model to enable pre-processing; 2) threshold optimization and diversified score indexing to reduce the number of accesses to the inverted lists; 3) candidate selection to reduce the number of diversified scores to compute.

Through experimental evaluation using three datasets and comparing profile diversification with other scoring functions, we showed that it presents the best compromise between all requirements we have identified. Through a user survey, we showed that profile diversity was preferred in a majority of cases. We also showed that exploiting user feedbacks to tune diversity during query processing enhances the user satisfaction. Finally, our experiments showed that our optimizations strategies could bring significant gain in terms of response time both on content diversification and profile diversification scoring.

References

[1] C. Yu, L. Lakshmanan, S. Amer-Yahia, It Takes Variety to Make a World: Diversification in Recommender Systems, in: EDBT, 2009, pp. 368–378.

[2] Z. Abbassi, S. Amer-Yahia, L. V. Lakshmanan, S. Vassilvitskii, C. Yu, Getting recommender systems to think outside the box, Proceedings of the third ACM conference on Recommender systems - RecSys '09 (2009) 285`doi:10.1145/1639714.1639769`.

[3] C. Ziegler, S. McNee, J. Konstan, G. Lausen, Improving recommendation lists through topic diversification, in: WWW '05, 2005, pp. 22–32. `doi:10.1145/1060745.1060754`.

[4] S. Abbar, S. Amer-Yahia, Real-time recommendation of diverse related articles, in: WWW, 2013, pp. 1–11.

[5] M. Servajean, E. Pacitti, S. Amer-Yahia, P. Neveu, Profile Diversity in Search and Recommendation, in: WWW Companion, 2013, pp. 973–980.

[6] M. Pazzani, D. Billsus, Content-based recommendation systems, The adaptive web (2007) 325–341.

[7] D. Goldberg, D. Nichols, B. Oki, Using Collaborative Filtering to Weave and Information Tapestry, Communications of the ACM 35 (12).

[8] C. D. Manning, P. Raghavan, H. Schütze, Introduction to Information Retrieval, Cambridge University Press, 2008.

[9] G. Salton, A. Wong, C. Yang, A Vector Space Model for Automatic Indexing, CACM 18 (11).

[10] R. Agrawal, S. Gollapudi, A. Halverson, S. Ieong, Diversifying search results, in: WSDM '09, 2009, pp. 5–14. `doi:10.1145/1498759.1498766`.

[11] A. Angel, N. Koudas, Efficient Diversity-Aware Search, in: SIGMOD, 2011, pp. 781–792. `doi:10.1145/1989323.1989405`.

[12] H. Chen, D. R. Karger, Less is More: Probabilistic Models for Retrieving Fewer Relevant Documents, in: SIGIR, 2006, pp. 429 – 436.

[13] X. Zhu, A. B. Goldberg, J. Van, G. D. Andrzejewski, Improving Diversity in Ranking using Absorbing Random Walks, in: HLT-NAACL '05, 2005, pp. 97–104.

[14] S. Gollapudi, A. Sharma, An Axiomatic Approach for Result Diversification, in: WWW, ACM Press, 2009, pp. 381–390. `doi:10.1145/1526709.1526761`.

[15] S. Amer-Yahia, L. Lakshmanan, C. Yu, Socialscope: Enabling information discovery on social content sites, in: CIDR, 2009, pp. –1–1.

[16] S. Amer-Yahia, M. Benedikt, L. V. Lakshmanan, J. Stoyanovich, Efficient network aware search in collaborative tagging sites, VLDB Endowment '08 1 (1) (2008) 710–721.

[17] J. Han, C. Moraga, The influence of the sigmoid function parameters on the speed of backpropagation learning, in: J. Mira, F. Sandoval (Eds.), From Natural to Artificial Neural Computation, Vol. 930 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 1995, pp. 195–201. `doi:10.1007/3-540-59497-3\_175`.

[18] A. Anagnostopoulos, A. Broder, D. Carmel, Sampling Search-Engine Results, in: WWW '05, 2005, pp. 245–256. `doi:10.1007/s11280-006-0222-z`.

[19] E. Vee, U. Srivastava, J. Shanmugasundaram, P. Bhat, S. Amer-Yahia, Efficient Computation of Diverse Query Results, in: ICDE '08, Ieee, 2008, pp. 228–236. `doi:10.1109/ICDE.2008.4497431`.

[20] Z. Chen, T. Li, Addressing diverse user preferences in sql-query-result navigation, in: SIGMOD '07, 2007, pp. 641–652.

[21] S. Amer-Yahia, J. Shanmugasundaram, Efficient Online Computation of Diverse Query Results, US Patent.

[22] K. El-Arini, G. Veda, D. Shahaf, C. Guestrin, Turning Down the Noise in the blogosphere, in: KDD, ACM Press, 2009, pp. 289–298. doi:10.1145/1557019.1557056.

[23] E. Feuerstein, P. A. Heiber, J. Martìnez-Viademonte, R. Baeza-Yates, New Stochastic Algorithms for Scheduling Ads in Sponsored Search, in: LA-WEB '07, 2007, pp. 22–31. doi:10.1109/LA-Web.2007.15.

[24] M. Drosou, E. Pitoura, DisC Diversity: Result Diversification Based on Dissimilarity and Coverage, Journal Proceedings of the VLDB Endowment 6 (1) (2013) 13–25.

AppendixA. Adaptive Diversity Loss Analysis

In this section, we analyze the quality and performance of the adaptive approach. Section AppendixA.1 proves Theorem 5.1 and Section AppendixA.2 proves Theorem 5.2.

In the two following sections, we use the set of notations presented in Table A.4.

AppendixA.1. Quality Loss Analysis

In this subsection, we show that given a keyword query q and L_q the set of inverted lists used to compute q's results, if q represents at least $(1-f_{max})\%$ of the queries that access each one of the inverted lists $L_k \in L_q$, then, the adaptive approach returns the same results as the basic algorithm ($R_a^q = R_b^q$).

To simplify the demonstration, we first consider *h-frequent* queries. Then, we show that if a query has a frequency of h on the inverted list it accesses, it is necessarily *h-frequent* if $h \geq (1 - f_{max})$.

Definition AppendixA.1 (*h-frequent* queries)**.** *Given a query q and L_q the set of inverted lists used to compute q's results R^q. The query q is said to be h-frequent if for each inverted list $L_k \in L_q$, the results of q that are in L_k are at least returned by $h\%$ of the queries that access L_k.*

Remark AppendixA.1. *If an item is accessed by $h\%$ of the queries, then its redundancy frequency cannot be higher than $1 - h\%$.*

First, we propose the following theorem:

Theorem AppendixA.1. *An h-frequent single keyword query processed using the adaptive algorithm has the guarantee to have the same results as it would have been processed using the basic algorithm ($R_a^q = R_b^q$) if $h \geq 1 - f_{max}$.*

Proof. Suppose a single keyword *h-frequent* query q where $h \geq 1 - f_{max}$. Suppose that q's results list R_a^q already contains $p-1$ items such that $R_a^q[1..p-1] = R_b^q[1..p-1]$. The p^{th} item of R_b^q is noted it. If $R_a^q = R_b^q$

Notation	Description
f_{max}	It is a system defined value representing a redundancy frequency. All items which redundancy frequency is inferior to this value have quality and performances minimum guarantees.
L_k/L_q	It is the inverted list corresponding to the keyword k or the set of inverted lists corresponding used to compute the query's q results list. if $q = \{k\}$ then $L_q = L_k$. $L_k[i]$ is the i^{th} item while $L_k[it]$ is the position of item it.
$R_{a/b}^q$	It is the results list of query q computed using the adaptive (a) or in the basic (b) algorithm
Δ	Given an inverted list, it's the variation between the indexation's score of an item in the basic algorithm and its indexation score in the adaptive algorithm
$rfreq(L_k, i)$	It is the redundancy frequency of the i^{th} item of the inverted list L_k.
$W(x)$	It is the weighed function that enables to decrease the indexation score of some items. It is monotonous and continuous decreasing
$1 - l_{max}$	It is the worst value $W(x)$ can have for items which redundancy frequency is inferior to f_{max}. $W(f_{max}) = l_{max}$
$th_{a/b}(s_1, ..., s_n)$	It is the threshold function used in the adaptive (a) or in the basic (b) algorithm
$sa_{a/b}(L_k)[i]$	It is the value of the sorted access performed on the inverted list corresponding to the keyword k at position i (using the adaptive (a) or the basic (b) algorithm's index).
it	It is an item
$rel(L_k, i)$	It is the relevance of the i^{th} item in L_k with respect to keyword k.
$nbSA_{a/b}^q$	It is the number of sorted access needed to compute R^q using respectively an adaptive index or an static index.

Table A.4: Notations used in the proofs.

then, the p^{th} item in R_a^q is *it*.

Recall that, in our approach, the items index score takes into account a weight that depends on the items redundancy frequency (*cf.* Algorithm 1):

$$sa_a(L_k)[i] = rel(L_k[i], L_k) \times W(rfreq(L_k[i], L_k)) \tag{A.1}$$

The threshold is evaluated based on the item's index score (*cf.* algorithm):

$$th_a(sa_a(L_k)[i]) = \frac{rel(L_k[i], L_k) \times W(rfreq(L_k[i], L_k))}{(1 - l_{max})} \tag{A.2}$$

Finally, the stop condition of our approach is the following (*cf.* algorithm):

$$\textbf{if } th_a(sa_a(L_k)[i]) > rel(L_k[i], L_k) \times W(rfreq(L_k[i], L_k)) \tag{A.3}$$

It means that when this condition is satisfied, the algorithm stops and adds the best items accessed until now in the results list. Therefore, if the condition is satisfied before accessing *it*, the results list R_a^q wouldn't be identical to R_b^q.

At some point, if *it* has been accessed, we know that it will be added to R_a^q because it has the best score and R_a^q will remain identical to R_b^q.

However, if *it* hasn't been accessed yet, its position is necessarily after the last accessed item. Let's say that the last accessed item is the i^{th} item in L_k. Because scoring function is monotonous, we can define the next equation:

$$rel(L_k[i], L_k) \times W(rfreq(L_k[i], L_k)) \geq rel(it, L_k) \times W(rfreq(it, L_k)) \tag{A.4}$$

where $L_k[R_a^q[p-1]] < i < L_k[it]$. Since q is *h-frequent*, the next item *it* is also *h-frequent* and since $W(x)$ is a continuous monotonous decreasing function, if $h \geq 1 - f_{max}$ then $f_{max} \geq 1 - h$ and we can define the following equation:

$$rel(it, L_k) \times W(it, L_k) \geq rel(it, L_k) \times W(f_{max}) \tag{A.5}$$

Which leads us to Equation A.6.

$$rel(it, L_k) \times W(it, L_k) \geq rel(it, L_k) \times (1 - l_{max}) \tag{A.6}$$

Finally, using equations A.2, A.4 and A.6 we can define the following inequality:

$$th_a(sa_a(L_k)[i]) \geq rel(it, L_k) \geq rel(it, L_k) \times div(it|R_a^q) \tag{A.7}$$

Because $R_a^q \leq 1$. Since the item *it* is the next element that should be added to R_a^q, we know that it is not possible to find an item at position i where $L_k[R_a^q[p-1]] < i < L_k[it]$ such that

$$rel(L_k[i], L_k) \times div(L_k[i]|R_a^q) \geq rel(it, L_k) \times div(it|R_a^q)$$

and therefore its score is higher than the threshold until *it* is accessed. Therefore, *it* will necessarily be accessed and added to R_a^q which will remain identical to R_b^q. □

Theorem AppendixA.2. *An h-frequent n keywords query is guaranteed to have* 0 *loss in terms of quality if* $h \geq 1 - f_{max}$.

645 *Proof.* Let the aggregation operator for multiple keywords queries be the sum function. Then the relevance of an item is the sum of its relevance in each one of the inverted list and the threshold is the sum of each threshold in each inverted list. We are given a n keywords query q and its results list R_a^q knowing that there are already $1 - p$ items in R_a^q. The next item to add in the results list is noted it. If Theorem AppendixA.1 cannot be extended for n keywords queries, it means that it is possible to find in one of the inverted list an item at position i where $L_k[R_a^q[p-1]] < i < L_k[it]$ whose score is higher than the threshold. Since the threshold in each inverted list is higher than or equal to the next item it, the aggregation of the threshold is higher than the score of the next item it. Because it is the next item to add, it is also the item with the bet score. Therefore, no item can have a score higher than the threshold computed until it is accessed. □

Theorem AppendixA.3. *An h-frequent query q that has always been processed at a frequency of h where $h \geq 1 - f_{max}$ will be at least $(1 - f_{max})$-frequent.*

Proof. Suppose that the query q is submitted and is at least $(1 - f_{max})$-*frequent*. Then, its results will be optimal (*cf.* Theorem AppendixA.2) and will at least reflect the frequency of the query. Since all queries are at least $(1 - f_{max})$-*frequent* at first, then q will remain at least $(1 - f_{max})$-*frequent*. □

Remark AppendixA.2. *The frequency of a query is very sensible to noise when the system has just started*
660 *because the frequency is evaluated with respect to a very small number of queries. However, after a certain time, the h-frequency of the queries will get stable.*

Remark AppendixA.3. *It is possible to define values of f'_{max} minimizing the probability that an item of frequency $h > 1 - f_{max}$, at a given time reaches a frequency less than $1 - f'_{max}$.*

As a conclusion, we guarantee that if the query's worst frequency is at least f_{max}, its results quality will remain optimal. The experiments show that, with our optimization techniques, even the queries that do not respect this property remain of high quality (*i.e.* their quality loss is very small).

AppendixA.2. Performance Analysis

In this subsection, we show that given a keyword query q and *indices* on the inverted lists used to compute q's results, if q represents at least $(1 - f_{max})\%$ of the queries (and therefore is at least $(1 - f_{max})$-*frequent*) that access *indices*, it is guaranteed to perform at maximum the same number of index's accesses than the basic algorithm.

In the following, we show that the only cases that could imply our approach to perform more accesses on the indices than the basic algorithm are impossibles. More precisely, only two cases could incur such behavior. First, since we know that queries that are at least $(1 - f_{max})$-*frequent* are optimal compared to
675 the basic algorithm, if the position in the index of the last result is increased then, the number of sorted

accesses needed to build the results list would be increased. Second, the threshold can no longer stop the algorithm at a specific rank in the index while it could stop it with the basic algorithm.

About the first case, generally, items in an inverted list follow a *Zipfian* distribution. In other words, there are a few items with very high scores and a lot of items with very small scores. Based on this idea, we make the following assumption:

Assumption AppendixA.1. *An inverted list of items follows exactly a Zipfian distribution:*

$$f(k; S, N) = \frac{1/k^s}{\sum_{n=1}^{N}(1/n^s)} \tag{A.8}$$

Where N is the number of items in the inverted list and k the rank of the item.

Based on this assumption, we propose the following theorem:

Theorem AppendixA.4. *For all values of $f_{max} < 1$, it is possible to find a value of $l_{max} > 0$ such that it is not possible to find an item at least $(1 - f_{max})$-frequent whose position increases.*

Proof. The position of an item increase if the score of a lower scored item increases, and therefor whose position is decreased. This is impossible because, at the beginning, $W(x)$ has its maximum value possible and $W(x)$ is a monotonous continuous decreasing function. Thus, items score can only be decreased compared to their score at the beginning.

Therefore, the position of an item can only increase because its weight $W(x)$ decreases. In other words, given the set of items IT which are at least $(1 - f_{max})$-*frequent*, then it is possible to find an item $it \in IT$ at rank r in the inverted list, whose position is increased if it is possible to find a solution to the following system:

$$\left.\begin{aligned} f(k; S, N) \times (1 - l_{max}) =& f(k'; S, N) \\ k' \geq& k + 1 \end{aligned}\right\} \tag{A.9}$$

In other words, the solution is the position of an item whose position could be increased. If this position is higher than the number of items in the inverted list, then, it would be impossible for our approach to perform worst than the basic algorithm. By solving the system, we can obtain the following inequality:

$$k' \geq \frac{1}{1 - (1 - l_{max})^{1/s}} \tag{A.10}$$

Therefore, it is possible to choose specific values of l_{max} that are small enough such that $k' > N$ which would imply that it is impossible to find a item at least $(1 - f_{max})$-*frequent* whose position k increases.

Moreover, if $l_{max} \to 0$, then $k' \to \infty$ which implies that the probability that the position of an item, which is at least $(1 - f_{max})$-*frequent*, increases tends to 0. □

Assumption AppendixA.2. *In the following, we assume that we are in the worst case, which means that all items which are at least $(1 - f_{max})$-frequent haven't seen their position decreased compared to the basic* *algorithm. Thus implies that the number of sorted accesses to compute the results list using the adaptive algorithm is at least equal to the basic algorithm.*

Then, we propose the following theorem:

Theorem AppendixA.5. *If a query is h-frequent and $h \geq 1 - f_{max}$, then the threshold condition enables the algorithm to stop, in the worst case, on the same item as the basic algorithm.*

Proof. Suppose a keyword query q at least $(1 - f_{max})$-*frequent* and its results list R_a^q. Suppose that R_a^q already contains $p - 1$ items. The next item to add is it and has already been accessed by our algorithm. 705

Recall that the smaller a threshold is, the faster the algorithm stops (*cf.* Equation A.3). In the following, we show that the threshold's value used in Equation 24 enables with very high probability to stop on the same item as the basic algorithm.

Since $rel(L_k[i], L_k)$ is static, the highest value of the threshold is computed when $W(l'[i]) \simeq 1$ (*cf.* Equation A.2). In the following, for simplification, we suppose that $W(l'[i]) = 1$ which is the worst case for our algorithm.

Therefore, in the worst case, the difference between the basic algorithm threshold and our threshold follows this equation (according to assumption AppendixA.2 and equation A.1):

$$th_a(sa_a(L_k)[i]) = \frac{th_b(sa_b(L_k)[i])}{1 - l_{max}} \tag{A.11}$$

Since we are in the worst case (*cf.* assumption AppendixA.2), the number of sorted accesses of our solution is either equal or higher than the basic algorithm. The only case where it would be higher, is when the basic algorithm's threshold satisfy the stop condition while our solution's threshold doesn't. This leads to the following equation:

$$\frac{th_b(sa_b(L_k)[i])}{1 - l_{max}} > rel(it) \times div(it) \geq th_b(sa_b(L_k)[i]) \tag{A.12}$$

In the following, we compute the probability that an item satisfies such equation. This probability is noted $P(nbSA_a < nbSA_b)$. Given T as all the possible values for the threshold, $P(nbSA_a < nbSA_b)$ is computed as follows for one keyword query:

$$P(nbSA_a < nbSA_b) = \sum_{t \in T} \int_{t \times (1 - l_{max})}^{t} f(x)dx \tag{A.13}$$

where $f(x)$ is the distribution of the final scores of the items whose range is in: $[0, 1]$. $f(x)$ is a continuous function. Equation A.13 calculates the probability to find an item it that satisfies Equation A.12, taking into account all possible threshold values. For n keywords query, it is the intersection of this probability for each one of the keyword. Developing the integral of Equation A.13 leads to the following equation:

$$\int_{t \times (1 - l_{max})}^{t} f(x)dx = F(t) - F(t \times (1 - l_{max})) \tag{A.14}$$

where $F(x)$ is the primitive of $f(x)$. Since $f(x)$ is continuous between 0 and 1, $F(x)$ remains continuous between 0 and 1 and since $l_{max} \to 0$, then $t \times (1 - l_{max}) \to t$ and $\int_{t \times (1 - l_{max})}^{t} f(x)dx \to 0$. The finite sum of a set of probabilities which are equal to 0 is 0. This probability is for one keyword queries. For several keywords, the probability is the intersection of the probabilities of each keyword which is even smaller.

In other words, the probability follows Equation A.15.

$$\lim_{l_{max} \to 0} P(nbSA_a < nbSA_b) = 1 \tag{A.15}$$

□

Remark AppendixA.4. *Suppose that the previous probability's event happens, it would imply a unique additional sorted accesses. However if the event happens several times, it will provoke as many additional sorted accesses but the probability that this event happens is even less likely.*

As a conclusion we've shown that, in the worst case, the probability that our algorithm performs a few more index accesses tends toward 0.