Collaborative(ly) Personalized PageRank

Master Thesis (Masterarbeit)

Théo Dubourg

September 15, 2014

Advisors: Dipl. Inf. Wölfl (Universität Passau), Prof. Dr. Calabretto and Dr. Laporte (INSA Lyon)

Reviewers: Prof. Dr. Brunie (INSA Lyon), Prof. Dr. Kosh and Prof. Dr. Granitzer (Universität Passau)

Contents

Abstract

This Master Thesis proposes a new technique to improve web search results by personalization. Indeed, we exploit the history of clicks and search queries as a support to personalize the computation of the PageRank score, which is then used during the final ranking of the web pages on the Search Engine Result Page (SERP). The objective is to study whether incorporating such usage-based information using collaborative filtering techniques and using PageRank personalization provide an improvement or not. According to the study of previous works in terms of web search personalization, recommender systems and PageRank computation, including PageRank personalization, no such works have been published previously. As a consequence, we contribute to the field here by providing a detailed, reproducible and rigorous study of how this technique can be implemented and how well it can work in the case of our implementation. Throughout our study, we will first review previous related works on topics such as context-aware systems, user modeling, recommender systems, web search personalization and the use of PageRank in personalized web search. Second, we will describe the new system we propose here to take advantage of usage data and integrate it into web search ranking via Personalized PageRank. Finally, we will describe the experimental setup we used to implement and evaluate such a system and the results we obtained. After analyzing such results and comparing them, when possible, to previous works' results, we will then provide a brief conclusion about the proposed system and its application and also point to future work that can be done on this topic.

Abbreviations

- **SERP**: Search Engine Results Page
- **PR**: PageRank
- **PPR**: Personalized PageRank
- **CPPR**: Collaboratively Personalized PageRank
- **IR**: Information Retrieval

Part I

Introduction

1 Introduction and motivations

1.1 General Context

The World Wide Web, based on the HyperText protocol, has been one of the most disruptive technological change in the recent past. The amount of information that is accessible through the WWW has become incredibly huge. IBM predicted in 2006 [1] that by 2010, the amount of information available would double every 11 hours. People also use more and more this available knowledge. While the WWW was, a couple of years ago, mainly used by computer scientist or people that were somehow working with computers, the mobile trend is bringing additional people online everyday.

While a lot of these new people come online to connect with peers, an incredible amount of them also come online to get access to additional knowledge. Using a search engine such as Google, Yahoo! or Bing has become the daily routine, not to say the hourly routine for many.

But while knowledge is *accessible*, it does not necessary means that it is *reachable*. In other words, people might only be a couple of clicks away from the exact information they need, but still unable to reach it because they have no directions to go in.

Search engines thrive to change this fact, but while their progression has been impressive in the beginning, they soon hit the limits of standard techniques in *information retrieval*.

One of the limits does not actually come directly from the *techniques* themselves but from the way human beings deal with looking for information. Human beings are constantly *in context* and naturally use this context to give additional meaning to every information they deal with. For instance, if you ask someone in a shop how much a given item costs, and she answers to you "5", you will automatically understand that this means *"5 euros"* because the current context defines for instance the currency, and the item you asked about clearly is more expensive than 5 cents, for instance.

When they query search engines, human beings work the same way, they express what they think their *information need* is. But in practice, previous work have shown that most of the queries performed on web search engines can be considered as “ambiguous”. What “ambiguous” here means is that although we might have all the technology necessary to retrieve the information the user needs, the way users expressed their information need is not precise enough for us to reply with the exact information they are asking. In other terms: we are missing the user context. In their most dramatic form ambiguous queries can have completely opposite meanings depending on what the context is. The renowned example of “Java” in computer science shows it well: Java is both the name of a programming language and the name of a touristic island. The documents containing the information the user needs when asking for “Java” have close to no chances to be the same in both contexts.

With this in mind, a solution would be that instead of expecting the user to disambiguate the query by herself, to understand the context in which the user is querying the web search engine in order to personalize her search. Such systems are called *context-aware systems*.

Among the first to talk about search personalization were L. Page and S. Brin in 1998 when they first wrote about the PageRank algorithm [2]. PageRank algorithm’s goal is to differentiate pages that have all close scores in terms of relevancy to the user’s query. As such, they suggested that the PageRank algorithm, which at that time was seen as a way to tweak the already-computed scores of the relevant pages, could be personalized to a “user’s point of view”. They tested at first by creating a PageRank from a “web page’s point of view” (as this was easily done using PageRank implementation they had at that time) and obtained significantly different top results depending on whether the web page they used was talking about Computer Science or Sports, for instance [2].

As a consequence of this experiment, they suggested that future work could lead to personalization of the PageRank using information directly coming from the user, such as bookmarks or frequently/recently visited web pages. Those URLs, coming directly from the user would make the PageRank consider “importance” of web pages against the view of the user according to what the user browse most often.

Search personalization is an example of a *context-aware system*. Research in search personalization has gone a long way since Brin and Page talked about it and many different techniques now exist.

Outside of the problem of *user modeling*, search personalization techniques propose different approaches to personalizing the results to the user. Some of them will target the categories the user is interested in, some other will use information from similar groups in order to provide the user with content she would not reach else.

We are contributing to the field with a new one that uses information from previous user behavior by using the logs of the web search engine in order to alter the way we apply PageRank to the web graph and as a result, alter the results that the end user will receive.

1.2 Research Questions

As discussed previously, *context-aware* systems need context. In the field, the context that is tied to a user is generally called the *user profile* and is based often based on a *user model* that describes what we want to integrate in every user profile in order to provide the application with the necessary information to provide personalized services.

This brings the first task of the thesis.

The User Model:

- Representing the user interests
- Enabling PageRank personalization
  - Related to specific nodes of the graph (link, URL).
  - With a scoring function that ranks the level of interests of the user for this node.

Having a good user model in hand is unfortunately not enough. Indeed, as it has been studied in the past, the vast majority of users a reluctant to providing context such as user interests by themselves [3].

One way to overcome this issue is the use the information about the user interests that we can *infer* from their actions instead of asking them to explicit them.

Several ways of doing this have been proposed in the past. In the case of web search personalization, two sources of user interests are generally pointed at: browsing history and/or search/clicks logs.

Search/clicks logs actually integrate partial browsing history information. Indeed, if a user issued a given query to the web search engine, she obviously visited the search engine page in order to do that, which is a first browsing history information. The second browsing history information we get comes from what is called the *dwell time* between clicks on the SERP (Search Engine Results Page). We can infer from there, with an acceptable certainty [4], whether the user was actually satisfied with the link he clicked on or not.

In our research, we will here consider search/clicks logs as the source of information. Which brings the second task.

The Usage Mining:

- Using search engine logs in such a way that they provides necessary information for PR personalization

Having mined the data, we need to turn it into something that makes sense in a *collaborative* way. Indeed, in this work we want to achieve *group* personalization where you gain advantage of the other users that are similar to you in order to improve your experience. We decide to position our work this way because single user personalization has proven to be less efficient [5] and can also only improve results on queries that the user would have already issued in the past.

The *collaboration* is going to be our third task.

The Collaboration:

- Representing users similarity with each other
  - Defining a scoring function that ranks similarity of two given users on a scale
- Comparing users between them
  - Modeling profiles such that they allow such similarity computation

- Exploiting similar users' interests to improve personalization of PR
    - Defining a group-based PageRank personalization score

Finally, the last task our work is going to be about is to use the graph structure of the web as final means of applying the search personalization re-ranking based on the previously collaboratively-computed scores. For this, we will use PageRank computation personalization.

The Computation Personalization:

- Using PageRank in such a way that personalizing it provides personalization to the results pages

The main objective of this master thesis being to answer whether or not using PageRank as a means of collaborative web search personalization provides good results, we can say that we attempt to answer the following research questions:

1. *What data from the clicks logs should we use to provide personalization to the end user through PageRank?*
2. *How should the user profile be constructed to allow collaborative personalization of the SERP through PageRank?*
3. *How well can the personalization of a graph-based scoring provide personalization to the end user?*
4. *How can one evaluate web search personalization in the case of search results re-ranking through PageRank?*

1.3 Contribution

In this master thesis, we design a model and a personalization system based on the analysis of usage. The main idea of the personalization system is to exploit the history of clicks as past interactions of users, using web search engines logs. The second main idea is to personalize a web-graph-based ranking by using the click logs information to attach personalized scores to nodes of the graph.

In our approach, we chose to work with PageRank as the web graph ranking algorithm for several reasons: first, it is the most broadly used, second, it directly provides means of personalization using the *personalization vector* that is part of the original PageRank formula [2], and finally, PageRank personalization has proven to be effective in the past [6].

The main components of the personalization system that we present here are:

- Search/clicks logs processing and data gathering and enrichment (chapter 7)
  Here we deal with extracting the data from a simple search/clicks logs. Additional processing will be needed and sometimes the logs will also need to be enriched before being fully usable in the context of our personalization techniques (section 12.4).

- Users collaboration based on their history of clicks and the semantic relationships of the their queries (chapter 7)
  We define a similarity function between users that takes both into account the semantic distance between users and the clicks they issued in the past.

- Collaborative PageRank Personalization based on collaboratively personalized scores of web graph nodes
  We execute a personalized computation of the PageRank by using its *personalization vector* together with a set of collaboratively personalized coefficients for the nodes of the graph for which we have been able to extract users past clicks.

- A user interface to allow for *user study*-based evaluation as in previous works [6] [7]
  We build web UI to allow user studies to be performed and to allow users to easily give formal and sound preferences towards a ranking versus the other one as well as naturally and transparently expressing their precision judgment of the displayed rankings (subsection 18.1.1).

1.4 Thesis Organization

In the remaining parts of the current manuscript, you will find the following parts:

- A state of the art
  Part II will provide a review of the previous related works and provide insights on what has already been attempted and succeeded in the past and what are their limits. The state of the art will be divided into two majors parts: Personalized/Recommender Systems and Personalized PageRank. The earlier will review different techniques of user profiling as well as recommendation and personalization systems themselves. The latter will review previous works on techniques to personalize PageRank scores and their applications.
- Our contribution to the field: Collaborative(ly) Personalized PageRank
  Part III will provide the details of our work, divided into two main topics. We will first discuss the *usage mining* side of the work and how data is being processed and handled all the way from raw clicks logs to web graph nodes scoring based on semantically similar users past behaviors. We will then explain how those scores are integrated into the PageRank computation.
- A thourough experimentation and evaluation review
  Part IV will finally present how we actually implemented such a personalized system. As with the rest of the work, we will emphasize on reproducibility of the current work and provide everything that is needed for future work in the domain and/or to do the experiment again in the first chapters of Part IV. The last chapter (chapter 18) will present our evaluation methodology and present an analysis of the experimental results that we gathered.

The dissertation will then end with conclusion about the presented work, how the research questions have been answered and discussion about future work.

Part II

Related Work

2 Introduction

In this part of the work, we are going to review and present previous works that are related to what we are working on.

More specifically, we will first briefly review general work on *context-aware* systems in chapter 3 as *search personalization* is a type of *context-aware* system. But we will not go into details as we are going to quickly focus on *personalized search* in chapter 4 and will more specifically review it from the point of view of *recommender systems*. We will first review how *user profiling*, which is a type of *context modeling*, has been done in the past in recommender systems and personalized search. We are not going to do a fully detailed state-of-the-art on personalized search for the simple reason that this field is a huge research domain and it would be both impractical and also partially irrelevant to review works on all different types of *search personalization*. Finally, the very last section of Part II will focus on previous works that have been conducted on *personalized pagerank*.

3 Context-Aware Systems, Context Models

"Context-aware" term was first introduced, together with context-aware computing, by Schilit and Theimer [8] in 1994.

By the time Anind K. Dey and Gregory D. Abowd [9] wrote in 1999 about context and context-awareness, they were already able to find about a dozen different definitions of *context* in the literature.

While context was originally perceived mainly as a matter of location and time, context has then been considered not simply as a state but as a part of a process in which users are involved [10].

Anind K. Dey and Gregory D. Abowd [9] tried to give a more general and formal definition of context: Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.

While this is a handy definition to be able to tell what is or is not a context, this keeps things quite abstract and not especially formal in terms of manipulation of this context by an application.

[10] tries to give another general definition of a context while being closer to what an application can actually manipulate: "context is about evolving, structured, and shared information spaces, and that such spaces are designed to serve a particular purpose".

As a consequence of the variety of context definitions, is the variety of definitions of a *context-aware system*, also named a *context-aware application.* Once more, we will take here the definition given in [9] as follows: A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.

We prefer this definition over more specific ones as, for instance, if in the case of search personalization the user's task is to retrieve some sort of information, to fill her information needs, user's task might change in the case of a recommendation system where the user might not especially be filling in her information needs but just acting for some sort of other activity as filling in a form, writing a message in a web page and so on. We might still want to be able to define what a recommendation system would be in such context. A more specific definition of a context-aware system, such as a system that would for instance take into account user's location, time, date and currently approximated information needs, would not allow us to define a recommendation system outside of the information retrieval activity anymore.

The idea of information spaces in the previous definition of context brings us to the need of a context model. A context model is used to define the enclosing environment of some system under study. In other words, the context is the surrounding element for the system, and a model provides the mathematical interface and a behavioral description of the surrounding environment.

[8] defines a context model as the container that will basically allow a context-aware system to take action based on its content.

[8] and [11] review a lot of different data-oriented context models and agree on the fact that they can basically be categorized as follows:

- Context as a matter of channel-device-presentation.
- Context as a matter of location and environment.
- Context as a matter of user activity.
- Context as a matter of agreement and sharing (among groups of peers).
- Context as a matter of selecting relevant data, functionalities and services (data or functionality tailoring).

Models of the first category take the application as the subject of study. Their main advantage is to have a centrally defined context that can be automatically learned and updated and a variable context granularity. But they have limited flexibility in terms of subject to study and in the end are often quite low-level which makes them well suited for a limited scope application only. CoDaMoS [12] and SOCAM [13] can be classified in this category.

Models of the second category generally provide space and time management, based on a centralized context definition too. They have the advantage of allowing information quality management and disambiguation (sometimes) which makes them a bit more flexible. They unfortunately generally have very limited context reasoning. “CASS” [14] is an example of such systems.

Models of the third category focus of “what the user is doing”. They generally provide with context history, thus allowing context reasoning, which are their main advantages. They are still generally centralized. Automatic learning of user activity from sensor readings is another interesting technique sometimes employed in such systems.

The fourth category is focused on groups approach and the problem of reaching an agreement about a context shared among peers. As such, the context definition is here distributed. Sophisticated time and location information is needed in order to reach the goal of such systems. In order to provide context reasoning or context quality monitoring, they however have to get rid of the low level issues and provide a high level of formality which is often quite hard to implement. CoBrA [48] could be seen as an example of this category.

The last category focuses on how the context determines which data and/or application features. To achieve such goals, time, space and user profile are commonly present and have to be highly developed in order to allow context reasoning and as a consequence decision making, to choose which data or application features need to be enabled or disabled. ConceptualCM [52] is an example of this category.

4 Recommender Systems and Personalized Search

4.1 Personalized Search As A Recommender System

In this section, we are going review the literature about two topics: Recommender systems and personalized search. The reason we are grouping these subjects together is because we believe they are fundamentally the same thing. More specifically, personalized search is in our opinion a type of recommender system.

Indeed, the role of a recommender system is to select, from a huge set of items, which ones are best suited for a given user. The most known and simple recommender systems are probably the movies recommendation systems or Amazon.com's items recommender system. In such systems, among a huge set of items (all of the world's movies, all Amazon's items), the system's role is to select a given sequence of *top-suited items* that the user will be presented, as the user cannot deal with the entire set of items and select from them by herself, because there are two many of them.

In its simplest form, and as it appeared first in the literature, a recommender system is based on *ratings* of items by users of the system. Users rate items that they have watched, listened, read or used. The role of the recommender system is basically to *guess* what rating the user would give to an item that the user has *not rated yet*. Let us call $r(u, i)$ the rating given by the user u to the item i, then the recommender system should *extrapolate* the value of $r(u, j)$ where j has not been rated by u yet.

Personalized search is nothing different from this. In personalized search, the set of items can be thought as all documents, or web pages that have been returned by the information retrieval engine. In most cases and for modern search engines, the number of those documents is definitely huge[1], and too big for the user to make a selection by herself. Thus, the system has to select the *top suited items* out of all the items that are similarly relevant to a query, and present this sequence of items to the user.

Moreover, a *rating*, in the case of personalized search, can be thought as the *relevancy* of a given item to a given query string and a given user. If users were to rate every web page that they read, the personalized search problem would become exactly similar as the one of guessing the rating a user would give to a given web page she has not rated yet. In information retrieval terms, a *guessed rating* $r(u, q, p)$ (u is the user, q the query, p the web page) becomes in our case the probability that the web page p is going to fulfill the *information need* expressed by u via the query q. Of course, users do not rate all web pages they visit, but users do not rate all movies they watch or all books they read either, thus this is not any different from the standard problem of recommender systems.

As we will see, the relationship between personalized search and recommender systems does not stop there. Personalized search works in the literature definitely made use of techniques from recommender systems and can easily be classified as such.

As a consequence, we will approach the literature review of personalized search systems from the point of view of recommender systems.

4.2 User Profiling

4.2.1 Introduction

A *user model* can represent very different sets of data. It can go from a pure *context model* as we discussed it previously, for instance broadly defined, trying to gather as much information as possible, to a very restricted and focused model,

[1]"pizza" on Google.com returns $203,000,000$ results as of August 21^{th}, 2014.

for instance *user interests* models, where the only important information is the distribution of probabilities of interest over a set of categories for instance. This last case is generally called *user profiling.*

As we already briefly discussed *context modeling* more generally in the previous section (chapter 3), we are going to focus in this section on *user profiles* that have been used in recommender systems and/or personalized search.

We are going to review previous literature grouped by the following criteria:

1. What is the content of the user profile? What data does it track?
2. What is the representation of the user profile? What structure is used to store it and what advantage does it have?
3. What data is it based on?

As part of 3. we will also be able to discuss explicitly provided data vs. implicitly collected data. As part of 2. we will also be able to discuss dynamic vs. static profiles and adaptive vs. static profiles.

4.2.2 User Profile Content

The user profile content is generally composed of two types of elements: basic elements and/or composite/aggregated elements.

The first case can for instance be seen in [15] [16] [17] [18] [19] where the profile content is composed of keywords. Other basic elements can include *queries* themselves, in the case of personalized search or even clicks, as in [20]. Basic elements have the advantage of being easy to both manipulate and understand. However, if a profile is only composed of keywords, for instance, the limitation is then that profiles are prone to ambiguous keywords. Similar profiles could in fact represent not so similar users because of the different meanings every user puts behind her words. One way to overcome this limitation is to provide semantics. Another way is to combine basic elements with composite elements (that can also bring semantics) in order to disambiguate them.

Composite elements are enriched or enhanced elements, sometimes aggregation of basic elements. This can be clusters, as in [17] and [21]. But this can also more simply be categories. Categories are probably the most often used elements of user profiles in personalized search, profiles of categories can for instance be found in [22], [23], [24] and [20]. Note that in [6], profiles of categories are also used but this time not as the *user* profile but as the web page profile.

Other works also include "concepts", that are often close to categories in practice, as composite elements. Other composite elements that can be found in the literature are also rules, or assumptions. Rules and assumptions could for instance be "this user does not use this button" or "the program should execute this rule for this user", they can be found in [16] and [25].

Generally speaking, profiles that only include basic elements will be easier to manipulate and can be easily integrated in a variety of applications that will post-process the profile to achieve personalization. On the other hand, profiles including very elaborated information or information that has been processed heavily will be more suitable for specific applications. They have the advantage of providing a user profile that is more unique but this will also come at the price of less possible operations on the profiles themselves (like profile-to-profile cosine-based similarity).

4.2.3 User Profile Representation

The user profile representation comes along with its content to provide the application with *features*. A *feature* of a user profile is generally a measure or metrics, a value along a given axis (or "dimension"), that will allow the system to change its behavior depending on it.

While exotic representations have also been published in the past, the main three structures that generally support the user profile's data are: a vector, a hierarchy or an ontology.

Vectors are the simplest structure found to support the content of the user profile and also quite often a structure that works very well. Vectors are used for instance in virtually all works that store user profiles as distributions of probabilities over a set of categories, we can cite [22], [23], [24] and [20] for instance.

In all of those works, using different paths, the authors ultimately constructed a probability vector over a set of categories that represented the user interest in this specific category.

The experiments have proven that this technique works well:

- [22] managed an accuracy between 80% and 100% in guessing the category a user would have classified her query into.
- [24] was able to reduce the number of *user actions* needed for the user to fulfill her information need by 90%–105% compared to standard search engines with their project *Outride*.
- [23] achieved precisions varying from 60% with the worst tested algorithms to more than 80% in guessing which page the user would consider as "hot" or "cold" in the category she is currently browsing.
- [20] observed a significant 23% improvement in search results accuracy.

Hierarchies can be found in [25] and [17] where they are used in order to represent the relationship between the topics of interest that are present in the user profiles. It should however be noted that [17] constructs the hierarchy an interesting technique based on connected subgraphs. Every node being a topic and every edge being a relationship between a topic and another, they iteratively remove edges that are above a given threshold and keeps the resulting fully connected subgraphs as the topics of the given level of the hierarchy.

Hierarchies have the advantage of providing semantics where vectors generally will not. Quite often, they will also rely on data structures that allow for *dynamic* user profiles, that is to say user profiles that are updated while the application is running and the user is using it, in comparison to vectors that are generally computed once or at specific intervals. Hierarchies also allow to perform different treatments depending on the level of the information. In the case of personalized search, for instance, depending on the ambiguity of the query and previous click entropy measures on this query, one can decide to use the higher-level disambiguations categories or lower-level ones. One could also disambiguate by going through the hierarchy from top to bottom as the research session goes. Hierarchies also allow level-specific profile-to-profile similarity computation. Finally, many works have been done on hierarchies shape analysis and this can also constitute a feature, as used in [17].

On the other hand, vectors also have advantages. For instance, they generally benefit from a wider set of proven and robust mathematical tools such as average vectors, many sort of similarities functions (as in [20]), but also, representations: data visualization eases the work of researchers in understanding behaviors and results of algorithms, allowing for faster iteration and improvement of research works. Linear combinations are also often used, such as in [6].

To summarize it, we could say that hierarchies generally provide higher-level features and more dynamic profiles, and even *adaptive* profiling methods. Adaptive methods are methods that will act differently depending on the context and the current user profile. While vectors would provide a higher number of operability features: post treatment, mathematical operations, etc. ...

Finally, examples of ontologies as the structure of the user profile can be found in [19] and [26]. Ontology have not widely been used yet and their use is quite often close to the one of a set of hierarchies, but with a semantic meaning to different levels and different hierarchies of the set.

4.2.4 User Profile Data Basis/Data Collection

The user profile data "basis" is what the user profile is based on. This can sometimes be straightforward data that is directly included in the profile but sometimes it can be pretty different from what ends up in the final profile.

We will differentiate mainly two types of data collection: explicit or implicit.

In explicit data collection, either the users are asked to explicitly provide the necessary data to generate their user profile or they are asked to provide data as part of another task and this data ultimately serves as the basis of the user profile generation.

Explicit data collection can be found in numerous works in the past, such as [25] [16] [26].

Explicit data collection has obvious advantages:

- There is no need for heuristics about what kind data should be used to generate what type of user profiles.
- The user profile *precision* is at its highest.

- The user can easily decide about what to provide and what not to provide in terms of data, thus avoiding privacy issues of data collection.
- If the user sees that the personalization/recommendation engine does not behave as she thought it would by providing such information, she can fix her profile in accordance.

To summarize: user control and precision are the main advantages. But one should also note that the *amount* of data necessary to generate a complete user profile will also likely be much lower. Indeed, in implicit data collection, we often need a significantly higher amount of data to generate a profile as we need to make sure the data is interpreted correctly by having confirming data. For instance, if a user clicks *once* on a link, we cannot really say whether she is satisfied with this link, but if she clicks 40 times on this link over a total of 50 clicks, then we are pretty accurate.

Unfortunately, explicit data collection also has significant drawbacks:

- It is generally costly.
- It is not automated (which goes along with costly).
- Users might still misunderstand questions asked by the system or how to use it and could provide completely wrong information.

But the main drawback remains the following: users simply do *not* want to provide explicit profiles/preferences as it has been shown in the past [3]. Thus, in addition to being costly, such explicit data collection will simply not be applicable when it comes to real world users.

Implicit data collection, on the other hand, has significant advantages, basically where explicit data collection has drawbacks:

- It is automated, and thus can be scaled more easily without necessarily leading to huge costs.
- It can be transparent to the user. Indeed, it can be done on data the system already collects in order to fulfill its task anyway.

- It reflects the *real* usage of the users: how they use the system in the *general* case, not in the special case of being explicitly asked question about this or that.
- Significantly higher amounts of data can be integrated in a given user profile as a consequence of the process being automated.
- The profile can be updated automatically without the need of any user action.
- Multiple profiles can be generated for a given user without necessarily multiplying the amount of resources or time necessary to generate them compared to having a single profile.
- As computer resources get cheaper and cheaper, automated processes get cheaper at the same time (while human resources tend to get more expensive over time).

The list can obviously be extended to all advantages and drawbacks of automated systems vs. human-driven/manual systems, but we think the point is clear with these few advantages highlighted. Past works using solely implicitly collected data include [20] [15] [17] [18] and [19].

Their most common drawbacks include:

- Precision and interpretation: One needs to build a model and transformation function from the collected data to the final user profile. Also, the collected data will generally be less specific and carry less information in the case of implicitly collected data compared to data provided directly by the user (example: X clicks over Y clicks vs. "I was satisfied with this link, I would rate it 4/5").
- Big data implications: as discussed in the introduction of the thesis about the *toxic terabyte* [1], modern systems tend to accumulate so much data that it becomes sometimes impossible to work with it.
- Privacy concerns: the collected data, and moreover its analysis, could reveal information about the user that she would not not want to be inferred

and even less to be integrated into her profile. While with explicitly collected data, she would have been able to simply not answer or provide false information the corresponding questions.

- Legal concerns: often a direct consequence of the privacy concerns, governments tend to regulate more and more the implicitly collected user data and it can sometimes become quite hard to legally collect enough data for the personalized/recommender system to function properly.

As we can see, while implicit data collection seems to still be the best choice in our opinion between explicit and implicit collection, both have advantages that cannot be ignored. This is one of the reasons why many past approaches to user profiling actually use both explicit and implicitly collected data, among the previous works that make use of both we can cite [25] [16] [26].

4.3 Applications in Recommender Systems and Personalized Search

Based on the previously described user profiles, recommender systems generate recommendations to the users.

When recommending a given item to a given user, recommender systems and personalized search can globally be classified as using one of the following combinations of sources of information:

1. Only user's profile and item's profile
2. User's profile and all (or a subset of) items' profiles
3. All (or a subset of) users' profiles and item's profile
4. All (or a subset of) users' profiles and all (or a subset of) item's profile

Depending on the sources used, recommender and personalized search systems are classified as follows:

1. Content-based recommender systems: Items will be recommended based on similarity to the ones the user already rated or preferred in some way in the

past. Such systems will use sources of information 1. and/or 2. in the above list.

2. Collaborative recommender systems: Items will be recommended based on what similar (tastes, preferences…) people preferred in the past. Those systems will use sources of information 3. in the above list.
3. Hybrid recommender systems: here, items will be recommended combining both methods. Those systems will use any of the sources of information previously described and sometimes even additional ones.

In [27], the authors use as similar classification of personalized search systems. They name the two first *content-based filtering* and *collaborative filtering*. In addition to these two categories, they also define two other categories as follows:

- Rule-based filtering systems ask the users questions. They then extract rules from the answers and then applies these rules to deliver appropriate content to the users.
- Web usage mining personalized systems provide personalization by applying data mining techniques to web usage data.

We believe, however, that rule-based filtering is just a specific case of content-based filtering. Indeed, in the end the system will use only the user profile (composed of rules) and the items profiles. And, in the end, the rules are used in order to recommend items but the items themselves are matched to the rules based on their content anyway. One could argue that *content-based filtering* is only about user profiles being composed of *contents*, but we prefer to classify recommender systems by their source of data as we believe this is what changes the most the way they work. There is indeed fewer difference between two systems that are both building a user profile - in two different ways - and items profiles and then make recommendations based on features of the user profile and features of the items profiles than there are between such systems and a collaborative filtering system, that will do user-to-user comparisons and base its recommendation on those comparisons rather than on comparisons to items profiles.

In the end, the classification of ruled-based filtering systems will depend, in our opinion, on the way the set of rules have been defined. If the set of rules is statically defined, we believe the system is just a special case of content-based filtering. However, in [28] for instance, the rules are themselves defined by detecting patterns in implicitly collected user data. In this case, an additional source of information is thus indirectly used at the time of the recommendation and this would classify itself as a hybrid recommender system.

As per *web usage mining*, the classification will greatly depend on what data is analyzed by the *data mining technique* that is used: Is the data collaborative? Is it based on items' content?

For instance, in [29], the authors describe their system as a hybrid recommender system, their technique integrates patterns and behaviors in the user profiles and they also analyze item-to-item relationships, which can basically be summarized as an advanced user profile and item content analyze and the use of both profiles to make the final recommendation.

In the two following sections, we are going to review in more details content-based filtering and hybrid filtering recommender systems and web search personalized systems. More specifically, we are going to focus on their application in personalized search re-ranking, as this is the main topic of our research. Re-ranking, as opposed to query rewriting, will apply a second algorithm on top of the IR scoring, in order to select the top set of items that are recommended to the user, among the set of returned items by IR engine. It is generally opposed to query rewriting, that we will not cover here, which is a technique where the query is generally enriched, or simply transformed, so that the IR engine returns results that are tailored to the current user. For each category (content-based, hybrid), we will present its specificities, advantages and main limits.

4.3.1 Content-based filtering: User-level Re-ranking

Content-based filtering can be observed in numerous previous works, such as [30] [31] [32].

It has been applied more specifically to web search personalization in for instance [6] [20] and more recently [33].

One of the main criticism that can be found against content-based filtering recommender systems is that, as their only source of information to estimate matching between a given item and the user's profile is the set of features of the items themselves, they are also inherently limited to items that are easily subject to automated parsing.

However, in the application of web search personalization, this limit is not necessarily an issue. If we limit the scope to web pages search (hypertext documents), such documents are very easily subject to automated parsing. And even hypertext features are anyway defined from the content of the web page and can thus be inferred by automated parsing easily. This makes this criticism towards content-based filtering not really relevant in the context of web search personalization.

A second main criticism is the general high sensibility of such systems to feature selection by the system's designer. Indeed , in the case of item recommendation problem, even if two items are not the same, if their profile against the set of selected features to match the user's profile is the same, such items will basically be recommended equally. In the case of web search personalization, however, the issue has much less impact. Indeed, the IR engine will take into account the whole document anyway and it is responsible for scoring documents based on their entire content. The recommender system comes in second position and helps rework those scores so that they fit better the user profile. Two items that are different enough, even if they have the same profile against the recommender system, will be differentiated at the first stage of the system, when the IR engine is run. Once more, this drawback has much less impact in the use case of web search personalization.

Where content-based filtering systems show their main limits, when applied to web search personalization, however, is when it comes to the *new user problem*, that we could also generally call the *cold start problem*.

What this problem means is that, as content-based systems only use the user profile, at one end, and items profiles at the other end, they will in fact only be able to *extrapolate* the utility of an item regarding a user (and regarding an information need, in web search) if the user already has a profile containing similar items (or similar queries). In the general case of recommender systems, this is mainly when a "new user" comes into the system. But in the case of web search, this problems appears every time the user makes a query unrelated to everything she has done in the past. For instance, if a user has a profile containing queries about surfing, tourism and travels, the recommender system will be pretty accurate differentiating items when the user will query for “java” between “Java Island” items and “Java Programming Language” items. But if this very user then queries for “mouse”, the system will not really have any accuracy in knowing whether this is about the animal or the computer device. Although “java” was already in the profile, it was disambiguated to not being related to computers and thus, we have no real information on what to recommend to the user for the query “mouse”.

This is where collaborative approaches and hybrid approaches will show their strength, as we will discuss in the next section.

4.3.2 Collaborative & Hybrid Filtering: Group-level Re-ranking

Collaborative filtering systems have had their moment of opulence, countless works have deployed variations of the collaborative filtering method in the past. Among them, [30] [29] [31] [32] are for instance popular examples. More recently, the approach was also benchmarked in [20].

In collaborative filtering, as explained in section 4.3, the source of data used for the recommendation or the personalization is not only the current user profile anymore, but all or a subset of users profiles. The main advantage of doing so is, in the case of personalized search, that it overcomes the *new query problem* where a user cannot be recommended anything if he has not already previously issued the query.

This is a significant advantage. Indeed, while *repeated queries* generally account for about 33% of the queries [20], which is significant, this would still mean that the personalization cannot be executed for 67% of the queries, which obviously significantly reduces its scope of application. Moreover, consider a user that has already issued an ambiguous query in the past: if she had low quality results and reformulated the query, she will probably directly query using the reformulated query the next time, especially considering the *auto-completion* mechanism now in place in modern browsers.

Collaborative filtering systems can generally be categorized as either *memory-based* or *model-based.*

The most common memory-based systems base their recommendations on a set of *similar users* to the current user. The similarity generally depends on the content of the user profile. It is generally defined as a function that takes as input two users, or two user profiles, and returns their *similarity*, respecting the standards of a *similarity function*, that is to say:

$$0.0 \leq sim(u_1, u_2) \leq 1.0$$
$$\mathrm{sim}(\mathrm{u}_1, u_1) = 1.0$$
$$\mathrm{sim}(\mathrm{u}_1, u_2) = sim(u_2, u_1)$$

The closer the similarity to 1.0, the more similar the users are, and the closest to 0, the less similar they are. This is not to be confused with a *distance function*, though, as this does not necessarily respects the triangle inequality, and $sim(u_1, u_2) = 1.0 \;\not\Rightarrow\; u_1 = u_2$.

Once the set of most similar users is defined, the system will then use a heuristic to generate an aggregated score out of the items scores against each of the profiles of the similar users. The most straightforward approach is for instance to simply average the item score over the set of similar users. This, however, gives as much importance to a user that is very similar to the current user and a user that twice less similar for instance. This can be overcome by using a weighted average using the similarity score as the weight, for instance. This is what is done in [20] for instance, and this is also what we will use in our work.

In [20], the score of a web page p is based on the ratio of clicks made on this web page compared to all clicks made for the given query:

$$score(q, p, u) = \frac{|Clicks(q, p, u)|}{|Clicks(q, \bullet, u)|}$$

where $Clicks(q, p, u)$ is the number of clicks that the user performed on web page p on the SERPs of the query q and $Clicks(q, \bullet, u)$ is the total number of clicks for user u for the query q.

Then, the collaboratively aggregated score is the weighted average over the set $\mathcal{S}_u$ of top similar users to u:

$$aggre(q, p, u) = \frac{\sum_{u_s \in \mathcal{S}_u} sim(u, u_s) \cdot score(q, p, u_s)}{\beta + \sum_{u_s \in \mathcal{S}_u} |Clicks(q, \bullet, u_s)|}$$

where β is a smoothing constant of 0.5 in the case of [20].

Model-based systems, on the other hand, generally consider the *rating guessing* problem from a probabilistic point of view. Indeed, one can consider that the recommendation score to be assigned to a non-yet-rated item for a given user is the *expected value* of a vote. [34] suggests to formalize it with the following formula:

$$E(v_{u,j}) = \sum_{i=0}^{m} i \times Pr(v_{u,j} = i | v_{u,k}, k \in I_u)$$

with $v_{u,j}$ the vote of user u for the item j and $E(v_{u,j})$ the expected value of such vote.

Based on this formula, model-based systems make their best to provide an estimation of the value of $Pr(v_{u,j} = i | v_{u,k}, k \in I_u)$.

Among previously used techniques, one can find a wide range of approaches: Bayesian networks are common, but neural networks and probabilistic relational models as well as linear regression and maximum entropy model can also be found in the literature [35] [36] [37].

However, when it comes to personalized search recommender systems, we have found no evidences of a purely model-based collaborative filtering system. Instead, most of them use a hybrid approach, combining memory-based and model-based approaches.

This is the case with [20] for instance. In [20], the memory-based component is using the number of clicks from the logs. But the final rating is not only an aggregation of ratings, it also uses the Bayesian classification of web pages into categories.

[38] is another example of a hybrid recommender system applied to web search. In this case the authors constitute *generative* models (from probability theory) on which they base their recommendation. But the generative model itself is based on clicks and documents contents.

[4] and [39] also use hybrid approaches to web search personalization, combining mathematical theory and memory-based principles of aggregation.

5 PageRank

5.1 Introduction

Numerous previous works have been done on PageRank personalization, among the most notable we can cite [40] [41] [42], but those works only focus on the PPR computation itself and how to make it more efficient, to scale it or to compute it differently from the standard PR computation presented in [2]. However, we found that only [2] and [6] studied ways to *apply* the PPR to improving web search results. Application of PPR have also been discussed, for instance in [7] and [43] but not applied to web search.

Web search ranking is not an easy task. Information retrieval techniques have gone a long way since the beginning of their application to the world wide web, even sometimes taking into account the context of a document to provide better ranking results, but they are still fundamentally limited by what they analyze: text, contents. The world wide web is a graph and as a graph that is being actively managed by individuals and constantly changing, it does not only provide *documents* but a whole *environment*. This environment contains for instance communities, it also contains hierarchies between pages, authorities, hubs, weak links and strong links, amongst other features. Those features can provide with far more relevant information on whether a given *page*, or *web document* should be ranked higher than another one or not.

It is especially true if two web documents have very similar - not to say equal - contents. In this case, IR-based techniques cannot provide any differentiation between the documents: those documents *are* the same, they are just not *located* in the same place in the WWW graph.

And while it might have been an edge case in the early days of the web, with the modern expansion of the WWW, documents that have equal contents are far more common.

This is more or less the original ideas behind the original PageRank work: How would one rank web documents other than by their content? Could we define an "importance" measurement that we could tag every unique web document (URL) with, so that even if two documents are the same we still have ways to differentiate them?

Since its publication in 1998 [2] PageRank has found its application domain ever expanding, as it basically is a quite robust way to give an importance score to nodes in many sorts of graph.

In this section, we are going to review the previous works that have been done in terms of PageRank applied to web search and more specifically in terms of *PageRank personalization*, as it was introduced in the original work [2]. We will not be reviewing here the numerous efforts that have been taken to optimize PageRank, tweak its algorithm to slightly improve either its runtime computation requirements or its precision. These works relate more to optimization directly related to the original algorithm and often to mathematical bases of the algorithm (including Markov Chains theory) than to an actual change in the way PageRank would rank the web graphs for different user profiles.

5.2 PageRank

S. Brin and L. Page presented in 1998 an algorithm they called "PageRank" [2]. PageRank was presented in the original work as an application of citation-ranking techniques used in research fields to *"bring order to the web"*, said another way provide an absolute score to every node in the web graph, allowing to order them in a unique ranking.

The PageRank algorithm is based on a "random surfer model". This model's goal is to provide a mathematically formal way to guess what pages a "perfectly random surfer" would visit. The assumption here is that if the number of *surfers* is high enough then the global behavior can be seen as one of a giant unique *perfectly random* surfer.

The important consideration in the *random surfer model* is that the surfer will basically perform two possible types of "move" on the web graph:

- Either she follows a hyperlink, from the web page she is currently viewing, to another web page that is linked by this page.
- Or she will randomly jump to another node of the graph, with a given probability.

There is not much to be said on the first type of move, but the second type of move provides us with an important ability. Indeed, if we are computing a *global*, *unbiased* and user-agnostic PageRank scoring vector, then we probably want to consider a uniform probability to jump to any node of the graph, as, taken globally, surfers could go anywhere.

But if we have some knowledge on the current user and we are computing a non-user-agnostic, said another way *personalized*, version of the PageRank scoring vector, we can tweak those probabilities to reflect the actual user behavior (as a surfer) and interests, thus taking into account the user in the PageRank computation.

Brin et. al. discussed this possibility in their original work. They use what they call themselves the "*personalization vector*" to compute a PageRank vector that is biased towards the "interests" described by the home page of a researcher's website. The obtained results are shown in Figure 1.

Web Page Title	John McCarthy's View PageRank Percentile	Netscape's View PageRank Percentile
John McCarthy's Home Page	100.00%	99.23%
John Mitchell (Stanford CS Theory Group)	100.00%	93.89%
Venture Law (Local Startup Law Firm)	99.94%	99.82%
Stanford CS Home Page	100.00%	99.83%
University of Michigan AI Lab	99.95%	99.94%
University of Toronto CS Department	99.99%	99.09%
Stanford CS Theory Group	99.99%	99.05%
Leadershape Institute	95.96%	97.10%

Figure 1: Page Ranks for Two Different Views: Netscape vs. John McCarthy (taken from [2])

Results show that the resulting PageRank is indeed dramatically different and the web pages that are the highest ranked are now web pages with strong relationships with the researcher's home page links.

We will not discuss further the mathematical/formal side of the PageRank algorithm in this section. Indeed, this will be discussed in details in section 9.1 and section 9.2. The *personalization vector* will specifically be discussed in details in section 9.2.

5.3 Topic-Sensitive PageRank

While [2] gave a first glance at how PageRank could be exploited using a Personalized PageRank to achieve search personalization, [6] was the first work done on actually implementing an algorithm of personalized PR.

[6] presents what they call "Topic-Sensitive PageRank" (TSPR). The authors divided the entire Web into 16 categories into which every web page should fit. The 16 categories are the 16 top-level categories of the ODP directory [44].

Each page is then computed its TSPR against each of the 16 topics. Each page thus has 16 TSPR scores, one for each of the 16 computed TSPR vectors instead of a unique score, corresponding to a unique PageRank vector. Each page is also assigned a probability to belong to one of those categories by using the class-probability of a naive Bayesian classifier with parameters to their maximum-likelihood.

They then integrate query-time information into the PageRank algorithm by looking at the keywords of the query. They extract the 4 most probable categories and use a linear weighted combination of the different TSPR vectors in place of the standard PageRank vector in the PR algorithm.

This approach has advantage over classical information retrieval (with PageRank) that no highly-ranked pages will appear in the top results *only because they are very popular*. For instance, for a query "java surroundings", Google currently returns a mix of Java island related results and Java programming language results. Considering the Bayesian classifier had access to the complete content of the page, pages about Java programming language will have a close to nil probability to belong to the "travel" class. As a consequence, when the classifier will classify "java surroundings" as a travel-related query, Java programming results pages will be filtered out.

[6] evaluates that the average precision of results on the queries that were tested by a group of volunteers changes from 0.28 with standard PR to 0.51 with TSPR. Unfortunately, their evaluation is a bit weak: A total of 5 volunteers were involved and 10 queries were used. As with [2], authors suggest to use more sources of context in order to compute the topics of the query keywords, such as session browsing history or bookmarks.

While it is unfortunate that the approach is a bit weakly evaluated, important considerations for this technique are that:

1. It reuses PageRank algorithm which has proven to be useful in breaking ties and relative ordering of search results.
2. Authors propose a way to compute it very efficiently, which still today is a concern with ranking algorithms.

5.4 Other Uses of Personalized and Topic-Sensitive PageRank

In this section, we briefly give directions to other works related to PPR.

In [43], the Topic-Sensitive PageRank is adapted and used to provide personalized search ranking inside folksonomies-based systems.

In [7], the authors use a website-locally-computed PageRank to predict the next pages that the user is going to visit. Their application domain is mainly browser and/or proxies pre-caching (proxies cache was also mentionned in the original PR work [2])

GFolkRank [45], Personalized SocialPageRank [45], SocialPageRank [46] and GRank [45] also use personalized PageRank or variations of PageRank, however their application is not web search either as they are primarily aimed at providing recommendation for GroupMe! [47]

5.5 Conclusion on Personalized PageRank

We have seen that Personalized PageRank is a subject that has already been discussed in the past. PageRank has a very wide range of applications and not all of the previous works actually apply to web search re-ranking, even though they are all context-aware systems.

Topic-Sensitive PageRank was especially close to what we want to achieve in our thesis: providing better web search results using the personalization vector of the PageRank algorithm to bias the computation towards a set of nodes that we believe have a higher relevancy to the current query than the average node. But the important limit of TSPR is that, while it is somehow bringing additional context by the use of a category classifier, the context is not brought directly by the user. Moreover, the work is limited to 16 pre-defined categories that are not based on the usage of the people querying the search engine but on an arbitrary dataset (the ODP directory [44]).

But the most important limits of the previous works on Personalized PageRank applied to web search are certainly the lack of automation, collaboration and semantical considerations.

In our work, we will focus on filling such gaps. We will do so by providing personalization based on automatically and implicitly collected usage data (click logs) and a collaborative approach (section 7.5) combined with a semantic clustering algorithm (developed in [21]).

Part III

Collaborative(ly) Personalized PageRank

6 Introduction

In this part we will present the novel approach that our work proposes in terms of using personalized PageRank and usage mining to provide web search personalization.

We will first present the logs-based usage extraction and how it allows us to compile user profiles.

Then we will explain how we apply collaborative-filtering techniques to these user profiles: how we define user similarity and how we use this similarity to define a unique score for web graph nodes, to be used in PageRank personalization.

The last section of this part III will go into the details of how the PageRank is being personalized using results of the collaborative-filtering scoring.

7 Logs-based Usage Mining

The two main goals of our usage mining here are the following:

- Extract user interests: General or more specific areas the user had interest in. Such as "computer science" or "sports." This gives general information on whether this user is more of a *computer scientist* for instance or more of an *accounting assistant* for instance.
- Extract user usage: Did the user, considering his interests, mainly preferred this or that link, when issuing requests of a given categories? Do users who query about "sports", when they make the query "fifa ranking", generally click on link A or link B?

The "user interest" here will help us match users together later on, while the "usage" will help us improve actual search results based on what users did in the past.

7.1 Logs Keywords Clustering

The first task in order to be able to say whether user A has interest in "sports" for instance, is to define what "sports" is. More generally speaking: in order to categorize users interests, we need categories.

Categorizing in the context of web search has been extensively studied in the past. Previous works on search personalization such as [20] sometimes reused categorizers from openly-held competition such as [48]. As the focus here is not on the *categorization* problem, that is often seen as an IR problem, we have decided to reuse a previously developed algorithm. This algorithm, presented in [21], has interesting properties that we do not find in most of the former categorizers.

Indeed, while most categorizers use IR techniques, and thus base their analyze of the actual content of a web document, [21] bases its analysis on semantical relationships, more precisely on taxonomies.

This has several advantages for us: the first advantage is that it does not require to analyze a huge number of web documents, avoiding the retrieving and storage costs. But the most important point in not using the web documents themselves is: all datasets currently available in terms of search engine logs are relatively old. Web documents will likely have changed contents or they might not even be available anymore. The second advantage is that it brings semantic meanings to the categories it produces. We will not go into details of all the differences between [21] and other clustering algorithms here. Another crucial point in the choice of this clustering algorithm was its availability: most previous research works are not openly accessible (or even not accessible at all), preventing from both proof of results and from reuse of the work. [21] clusters have been made available easily and in an open format that is easily readable.

Finally, the last interesting point of this clustering algorithm is that it has been evaluated on the same dataset as the one our thesis is going to be evaluated on. Which means we can refer to the previously published results in order to know how good it is on different parts of the dataset and the behavior of the algorithm is predictable.

The algorithm has, among others, a threshold parameter that will basically control the number of outputted clusters. We chose the 1.314 threshold as a recommendation from the author [21]. This breaks the entire set of keywords from the dataset down into 130 different clusters.

We also have to note that those clusters have been done on a post-processed version of the keywords set extracted from the AOL logs: A set of blacklisted keywords, mainly related to pornography, have been pruned. This is absolutely no problem for us. Indeed, we could not have used such keywords in the *user study* anyway, so the clustering provides us with a way to prune the set of available queries for evaluation: if a query has a nil clustering vector (nil weights for all the clusters), then it is thrown away.

7.2 Queries Clustering/Categorization

The keywords clustering brings to us the sets of *categories of interests* expressed by the entire set of queries and clicks from the logs. But we do not have a direct mapping from those to the users. The link between the users and those keywords are queries.

As a consequence, following the original goal of extracting user interests, the next task is to assign to every query from the logs, a *clustering vector*.

A *clustering vector* is a one-dimensional vector of as many coordinates as there are clusters. The i-th coordinate corresponds to the probability that the query belongs to the i-th cluster. This probability is calculated as follows:

$$c_i(q) = \frac{|kw(q, cluster(i))|}{|q|}$$

where $cluster(i)$ represents the i-th cluster and $kw(q, cluster(i))$ is the set of common keywords between q and $cluster(i)$. $|q|$ is the number of keyword of the query that also belong to the global set of keywords of all the clusters.

As a consequence, the *clustering vector* for the query q is defined as follows:

$$c(q) = \begin{pmatrix} c_0(q) \\ c_1(q) \\ \vdots \\ c_{130}(q) \end{pmatrix}$$

7.3 Logs-based Interest Extraction

The second main task in order to be able to say whether user A has interest in “sports” for instance is to actually define what “being interested in sports” means.

In our case, “being interested in X” means “the amount of clicks user A performed on X-related queries, compared to other queries”.

More formally, we define the interest of user u for the cluster $cluster(i)$ as follows:

$$int(u, cluster(i)) = \sum_{q \in \mathcal{Q}(u)} P(q|u) c_i(q)$$

where $\mathcal{Q}(u)$ is the set of queries from the *long-term history* of user u, $c_i(q)$ is the same as previously defined and $P(q|u)$ can be thought as the probability that user u will click on any result of the query q. It is formally defined using the *long-term history* of clicks of user u as follows:

$$P(q|u) = \frac{clicks(q, u)}{clicks(\bullet, u)}$$

where $clicks(q, u)$ is the number of clicks that the user performed on SERPs of this query and $clicks(\bullet, u)$ is the total number of clicks for user u. This formula is an adaptation of the work done in [20] but here we aggregate at the query level and not at the page level as our category clustering is done at the query level and not at the page level.

7.4 User Profile Compilation

The overall *user profile* can be thought at the definition of *user interests* over the entire set of categories.

Thus, we can define the user profile as follows:

$$c_l(u) = \begin{pmatrix} int(u, cluster(0)) \\ int(u, cluster(1)) \\ \vdots \\ int(u, cluster(130)) \end{pmatrix}$$

Although, following the idea in [20] we will consider that queries issued by many users are less important when building the profile, we will thus define an additional weight $w(q)$ as follows:

$$w(q) = \log \frac{|\sqcap(\bullet)|}{|\sqcap(q)|}$$

where $\sqcap(q)$ is the set of users who issued query q.

The $int()$ function thus becomes:

$$int(u, cluster(i)) = \sum_{p \in \mathcal{Q}(u)} P(q|u) w(q) c_i(q)$$

For convenience, we will write the user profile directly in function of the clustering vector of every query of $\mathcal{Q}(u)$. This gives the following vector equation:

$$c_l(u) = \sum_{p \in \mathcal{Q}(u)} P(q|u) w(q) c(q)$$

with $c(q)$ as previously defined (section 7.2).

7.5 Logs-based Usage Extraction

Now that we have the user profiles defined. What we want to do is to actually *infer*, from user profiles, *usage* information related to multiple users at the same time: *a group.*

First, for a given user u, we will defined the *neighborhood of* u as the set of the K-nearest neighbors to u.

The K-nearest neighbors will be computed based on a similarity function.

The similarity between two users u_1 and u_2 is defined as in [20], as follows:

$$sim(u_1, u_2) = \frac{c_l(u_1) \cdot c_l(u_2)}{||c_l(u_1)||\ ||c_l(u_2)||}$$

Note that, since $c_l(u)$ is a distribution over a semantic clustering of the queries, $sim(u_1, u_2)$ can be considered as taking into account the *semantic distance* between the two users. Indeed, $sim()$ is a vector product of two vectors defined in a "semantic vector space" (and vector space where axis represent semantic concepts/ features).

The K-nearest neighbors can then be defined as the K users with the highest similarity. In mathematical terms, this could be formalized as follows:

Consider the "sequence of ordered users" $\mathcal{U} = (u_0, u_1, ..., u_n)$, such that $\forall u_i \in \mathcal{U},\ sim(u, u_i) \geq sim(u, u_{i-1})$

Then the K-nearest neighbors is the sequence $\mathcal{S}_u = (u_n, u_{n-1}, ..., u_{n-K}), u_i \in \mathcal{U}$.

We can then see the *neighborhood of* u as describing a *usage group*, that is to say: a group of users all being representative of a given sort of *usage* of the search engine. Said another way, we could also consider such users as a *category of users.* But a given group is relative to a given user u, so $\mathcal{S}_u$ is not "all the users belonging to a given category", nor it is a "users clusters", it just is "a group of users with similar usage".

8 Web Graph Nodes Personalization Score

Now that we have a *usage* defined, we are going to score URLs of the web, web graph nodes, according to the usage group of the current user, thus providing a personalized score.

We take the definition of the score based on similar user profiles from [20], as follows:

$$score(u, q, p) = \frac{\sum\limits_{u_s \in \mathcal{S}_u} (sim(u_s, u) \cdot |clicks(q, p, u_s)|)}{\beta + \sum\limits_{u_s \in \mathcal{S}_u} |clicks(q, \bullet, u_s)|}$$

where u is a given user, q is a given query and p is a given page from the SERP of the query q. β is a smoothing factor that is equal to 0.5 here.

This way, the more similar a user is to the current user, the more impact the usage of the other user has on current user's own personalized score. Note that the usage of the user u itself does not appear anymore here. It is taken into account in the $sim()$ function and thus it impacts globally the computation, but the clicks data itself is coming from the similar users, this is why we call this a *collaborative filtering* approach: it uses the same principle as recommender systems using *collaborative filtering* algorithms.

9 PageRank Personalization

9.1 Recalls on PageRank Computation

As presented in the related work (section 5.2), PageRank is a recursively defined ranking algorithm whose goal is to provide a *absolute* ordering to *all* nodes of a *web graph*. It also has other applications and is not limited to web graphs, but it was introduced in this purpose. We reviewed in Part II how it is based on the *random surfer model* but did not go into the details of the PageRank computation. We will cover them now.

The basic idea of PageRank ranking system is that if page u has a link to page v, then the author of u thinks v has some importance. $stackoverflow.com$, for instance, is intuitively an important page, considering the high number of links from other nodes of the graph pointing to it. How does one determine the importance a page u will give to a page v? The most important the page u, the highest standard of contents and thus of links, is supposed to be on page u. Thus, the higher the *importance* u has, the higher the importance it *transfers* to page v by linking to it. This can be seen as an analogy of, indeed, academic citation ranking: if a paper is cited by a very important paper, it gains more importance than if it is cited by a newly released paper that has not proved to be very important yet (by being cited itself too). The (simplified) definition of the transfer of importance between u and v can be seen as follows [6]:

$$\forall v, Rank^{(i+1)}(v) = \sum_{u \in \mathcal{B}_v} \frac{Rank^{(i)}(u)}{N_u}$$

where v, u are pages (web graph nodes), $\mathcal{B}_v$ is the set of *backlinks* of v, that is to say the set of other pages that are linking to v and N_u is the number of *outlinks* of u, also called *outedges* for *outgoing edges* (remember the web graph is a *directed* graph).

Notice that in practice, the latter formula is expressed with a damping factor $c < 1$ that is used to balance the fact that there are nodes with no outgoing edges and their *importance* (also called *weight* later on) is lost from the system [2]:

$$\forall v, Rank^{(i+1)}(v) = c \cdot \sum_{u \in \mathcal{B}_v} \frac{Rank^{(i)}(u)}{N_u}$$

[2] also explains that the starting set of values $\forall u, Rank^{(0)}(u)$ does not impact the stabilized values, but it does impact the number of iterations the computation will take. As a consequence, "updating" a PageRank vector should be much faster than computing it for the first time. The values of the weights will indeed already be quite close to their real value, thus reducing significantly the number of required iteration. This makes PageRank a good candidate for web graph applications as typical web crawlers never stop crawling and are always updating the link database (as there is no real *end of the crawl* as the web likely expands faster than what you can crawl).

The previous formula is used for every node of the web graph and we repeat/ iterate it until the ranks converge to stable values [6].

But the way PageRank computation is generally handled is using matrices. Let $M = ((m_{uv})), (u, v) \in [[0, N-1]]$ be the *links matrix*, N being the total number of nodes in the web graph. Let $m_{uv} = \dfrac{1}{N_v}$ if there is a link from u to v, $m_{uv} = 0$ else. Now let $R = \begin{pmatrix} w_0 \\ \vdots \\ w_n \end{pmatrix}$ be the *PageRank vector* where w_u is the PageRank score for page u.

Then, we can write the previous formula for all web pages at once by writing, in matrix notation:

$$R = cMR$$

This equation means that R is an eigen vector of M with eigenvalue c. More specifically, PageRank uses the dominant eigenvalue of M [2], which can be computed by repeatedly applying M to any nondegenerate starting R vector, which confirms the previous characteristics about R starting values not impacting the final PageRank scores.

If the web were a fully connected directed graph, we could stop here. But a pattern in the web graph causes a problem: What happens when two pages u and v link *only* to each other. Now consider there is a third page p that links to one of them. This is what the authors called a *rank sink*: indeed, the "rank", or importance ranking, will be transferred to u and v through p but they will never redistribute what has been transferred to them, they will simply continue mutually grow their own rank without diluting it to the rest of the graph. This is problematic because the global "importance" should transfer ad-hoc to all the nodes of the graph at every iteration.

To overcome this problem, the original authors introduce what they call the "rank source" vector E.

The first presented equation becomes as follows:

$$\forall v, Rank^{(i+1)}(v) = c \cdot \left(\sum_{u \in \mathcal{B}_v} \left(\frac{Rank^{(i)}(u)}{N_u} \right) + E(u) \right)$$

This "rank source" vector will basically ensure that at every iteration every node of the graph will indeed receive a minimal amount of rank. The $E(u)$ function is originally a constant function of value $\forall u, E(u) = \frac{1}{N}$ with N the number of nodes in the graph.

The matrix notation becomes:

$$R = c(M + E)R$$

9.2 The PageRank "Personalization Vector"

We just defined that PageRank computation can be summarized as the following equation, in matrix notation:

$$R = c(M + E)R$$

where R is the *PageRank vector* that holds all the PageRank scores for all the web pages of the graph, M is the *links matrix*, that holds the linking information from and to every page in the graph, and E is the "source of rank" vector.

This is where it starts being interesting for us: The E vector is also called *personalization vector* by the authors. Indeed, when $E()$ is a constant function, E vector simply adds, in terms of the *random surfer model* (presented in section 5.2) a uniform probability to all nodes that the surfer will randomly jump to them.

But what happens if we decide $E()$ not to be constant anymore? Then we are *biasing* the computation of the PageRank towards certain nodes compared to others, because we are increasing the probability to randomly jump to those nodes at every iteration of the PageRank computation.

Note that the fact we are adding a little more chances to jump to those nodes at *every* iteration means this is not doable by simply post-processing the result of the PageRank, it needs to actually be done during the PageRank computation, which makes this technique quite expensive.

Using the *E personalization vector* is not something new that we are introducing here. It is not even something theoretical that has never been applied in the past: both the original work [2] and the latter "topic- sensitive" [6] version have applied this technique with success, which gives us a strong basis for our work. In [2] the authors simply ran a test of Personalized PageRank (PPR) computation by setting the home page of a researcher has the only node in the graph that had a non-null weight in the personalization vector. The result was immediate: the ranking completely changed and the top-ranked links were now links that were directly linked from the home page of the researcher and thus, supposed to be strongly related to the researcher's interest. As a consequence, the authors originally suggested that this could for instance be used by setting the personalization vector with non-null weights only on the URLs that would be in the user's bookmarks. The issue with this is, as we previously discussed, that users do not want to provide information by themselves, so we have to infer it, which is what we are doing in this work.

In the second work on PPR [6] the authors used a different approach about how to generate this personalized version of E.

As said earlier (section 5.3), they divided the entire web into 16 categories, taken from the ODP [44]. Then, they generated a PPR vector for each of those 16 categories. The E vector, for every category $\mathcal{C}$, is set as follows:

$$E(\mathcal{C}) = \begin{pmatrix} p_0 \\ \vdots \\ p_u \\ \vdots \\ p_n \end{pmatrix}, p_u = \begin{cases} \dfrac{1}{|\mathcal{C}|} & u \in \mathcal{C} \\ 0 & u \notin \mathcal{C} \end{cases}$$

where p_u represents the coordinate of the vector for the web page u and $|\mathcal{C}|$ is the number of URLs in the category $\mathcal{C}$.

The issue is that it means we now have not *one* personalization vector, but *a set* of personalization vectors. We are going to see in the next section how to combine them.

9.3 Combination of PageRank vectors

The original work on PageRank [2] has not explored ways to actually combine different PageRank vectors into a global vectors that would be composed of locally computed specificities. But the derived work [6] explored it.

As in the case of [6], the same way they did for web pages, they used IR techniques to provide every request with a specific probability to belong to one of their categories. They then use those probabilities as a weighting factor for the related PPR vector, building the global PR vector as a linear combination of the "categorized" PR vectors:

$$R_{combined} = \sum_{\mathcal{C} \in \mathcal{T}} P(\mathcal{C}|q) R(\mathcal{C})$$

where q is the current query, $\mathcal{T}$ is the set of all categories (16 in [6]), R_c is the PPR vector that has been computed using the personalization vector $E(\mathcal{C})$ and $P(\mathcal{C}|q)$ is the probability that query q belongs to category $\mathcal{C}$, where [6] uses a formula that makes itself uses of scores computed by a multinomial naive-Bayes classifier.

9.4 Usage-based PageRank Personalization

In our work, we use a *personalization vector* E defined using the collaborative scores previously computed in chapter 8.

The E vector is thus defined as follows:

$$E(q,u) = \begin{pmatrix} p_0 \\ \vdots \\ p_i \\ \vdots \\ p_n \end{pmatrix}, p_i = \begin{cases} \dfrac{1}{N} & i \notin clicks(\mathcal{S}_u, \bullet, \bullet) \\ score(u,q,i) & i \in clicks(\mathcal{S}_u, \bullet, \bullet) \end{cases}$$

where u is the current user, q is the current query and i represents a page. The $score()$ function is the one we defined in chapter 8 and N is, as in previous equations, the total number of nodes in the web graph.

As the scores being used in E have already been computed collaboratively, the *collaboratively personalized PageRank vector* $CPPR(q,u)$ can be defined as the PageRank vector computed using the personalization vector $E(q,u)$. This means the original PageRank equation, as presented in section 9.1 becomes:

$$CPPR(q,u) = c(M + E(q,u))CPPR(q,u)$$

The $CPPR(q,u)$ vector is the personalized PR vector we are going to use in our experimentation (Part IV).

Nevertheless, let us still have a look at what we would obtain if we were to apply the technique of weighted linear combination of PR vectors (section 9.3) to our work. It would give the following:

$$CCPPR(u, q) = \sum_{u_s \in \mathcal{S}_u} sim(u, u_s) CPPR(u_s, q)$$

where S_u is the previously defined (section 7.5) set of K-nearest neighbors of u and $CPPR(u_s)$ is the CPPR vector as previously defined. In this current case, such a technique would mean that we not only use the similar users to the current user, but we also use the users similar to the earlier similar users. Indeed, u_s here is a user similar to u, but $CPPR(u_s, q)$ is then computed using the personalization scores that take into account the users similar to u_s. This would thus integrate even more collaborative information.

However, in the current work, we are going to limit the *collaboration* to the CPPR vector level. We let the experimentation of a two-level collaboration (CCPPR) to future work (section 20.3).

Part IV

Experimentation: Implementation & Evaluation

10 Introduction

In this part of the work, we are going to present in details the evaluation and the experimentation that we performed on the proposed algorithm and personalization system. We will put an emphasis on results being reproducible as a personal choice as we believe a lot of published works are either experimented and/or evaluated on proprietary datasets or are missing crucial information that prevents reproduction of the results. Our entire implementation is also available online [49].

Part IV is going to be organized as follows:

- First, we will present in details the openly available dataset that we chose to use. We will present its characteristics and some statistics about it, allowing for comparison with earlier/later works' datasets.
- Second, we will explain in section 12.2 why we chose this dataset as the best candidate for our evaluation. We will also explain why the dataset is still not perfect and will present the "enrichment" (post-processing) process we choose to approximate missing information.
- The third section will cover the tools and models we used in terms of web search. How we generate the web graph that we are using to compute PageRank and how we store it and search it using standard IR tools.
- The fourth section will detail the PageRank computation, explaining how the heavy computation implied by the formulas presented in Part III is carried on.
- The last section of this part will present the evaluation measures themselves, the methodology and the results that we obtained with this methodology.

11 AOL Search Logs

The dataset we used to run our experiments is the AOL Search Logs that were released in 2006 under a non-commercial/academic license. These logs consist of 21,011,340 queries collected from 657,426 users over a 3-month time interval [50]. According to comScore Media Metrix, the AOL search network had 42.7 million unique visitors in May 2006, so the total data set covered roughly 1.5% of May 2006 search users for instance. The queries are also only taken from the ones that were issued through the *AOL Client*, meaning that the "user id" information is reliable. Indeed, it is based on users being authenticated and not based on cookies, which could lead to wrongly assigned ids or to the loss of an id in the middle of a search session. According to the dataset's original description[2], the queries are entirely unfiltered and revealed in plain text in the logs, which makes it a perfect candidate for personalization. The fact that the logs are *unfiltered* also means that most of the top queries are about subjects like sex or pornography, though, which makes a pre-processing necessary in case of users study, as we will see later.

11.1 Clicks Distribution

11.1.1 Of Users

As we can see in Figure 2 (where both axis' scales are logarithmic), the number of users for a given number of clicks first has a sharp exponential decrease in the beginning and tends to something more linear towards the end. This can also be seen in Figure 3 where we can also remark that 50% of the users have less than 11 clicks.

This could seem to mean that users mainly have very few information to give in terms of clicks to be included in their *user profile*, but in the end the average number of clicks per user if about 37, which already gives a reasonable amount of information.

It is worth noting that the dataset does not include any user with an empty profile. All users of the dataset have at least one click.

[2]http://www.gregsadetsky.com/aol-data/U500k_README.txt (last visited 14 Sept. 2014)

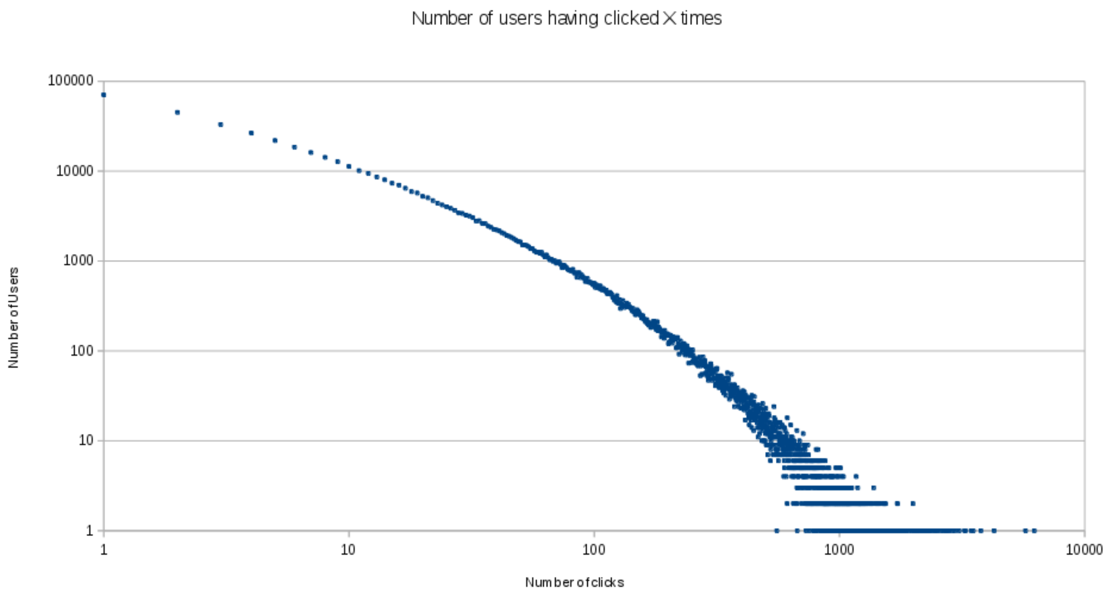


Figure 2: Clicks Distribution: Users with a given number of clicks

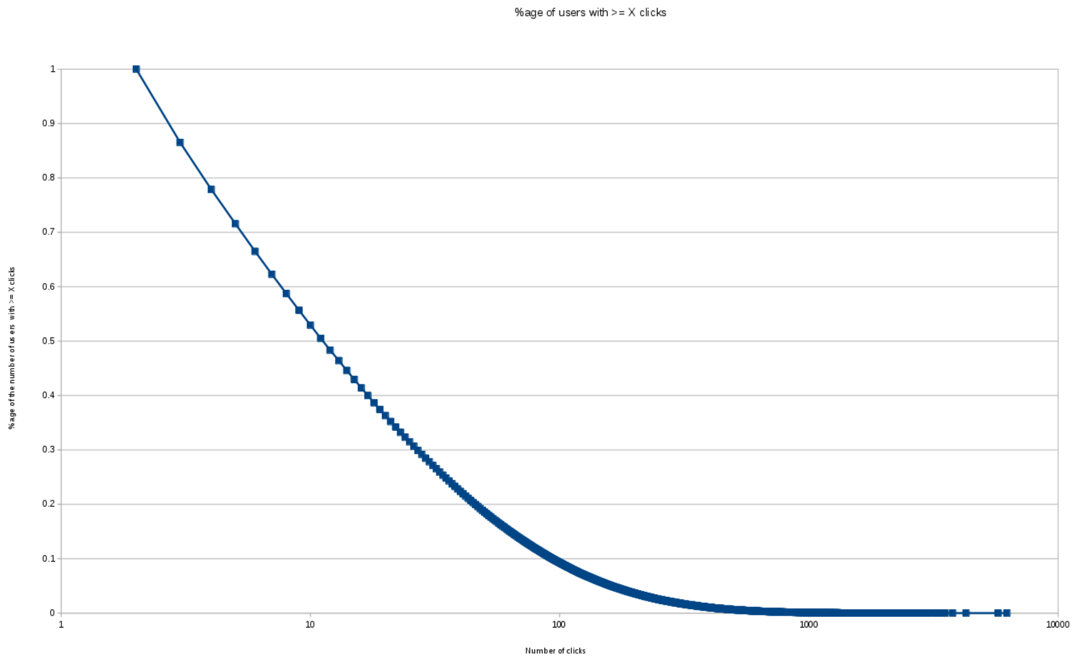


Figure 3: Clicks Distribution: Percentage of users with more than X clicks

11.1.2 Of Queries

The distribution of clicks per query string is shown in Figure 4. We clearly see that it also accuses a sharp exponential decrease, but this time, stays mostly stable.

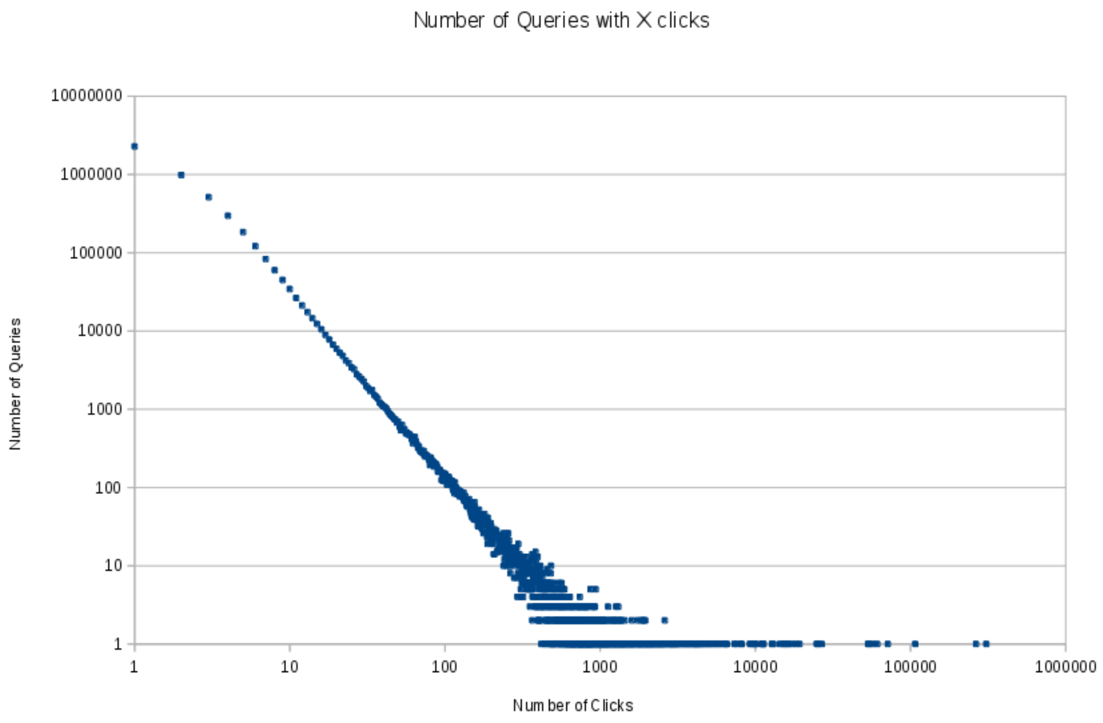


Figure 4: Clicks Distribution: Queries with a given number of clicks

It is interesting to remark that about 48% of queries have only one click. And only 5% of queries have more than 10 clicks, as shown in Figure 5.

The same way the dataset does not include any user with no clicks, we can see here that it also does not include any query with no clicks. AOL originally explained that the dataset was basically the dump of an internal database which itself was only a small sample of their search engine queries on the same period, but did not provide much information on how the sampling was done. Either AOL researchers database was originally sampled in purpose with only queries and users with clicks, or the researchers, which at that time released the logs with clicks analysis in mind, decided to filter out the queries or users that had no clickthrough information.

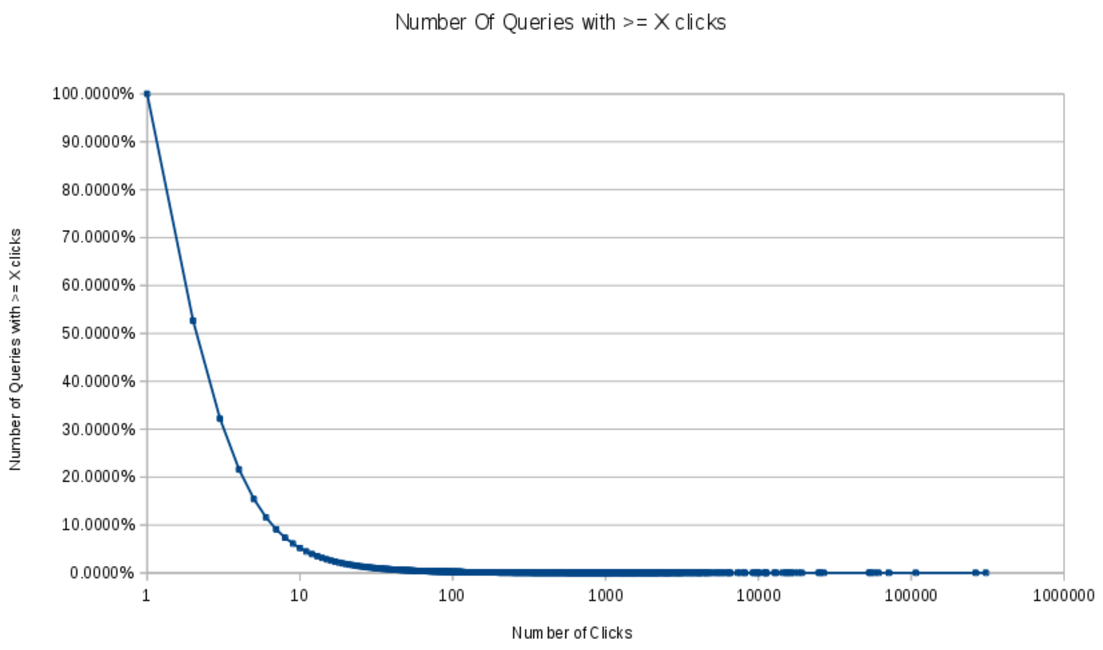


Figure 5: Clicks Distribution: Percentage queries with more than X clicks

11.2 Queries Per User and Clicks Average Distribution

The queries frequency distribution, that can be seen in Figure 6 and Figure 7 seem to follow roughly the same rules as the two previous distributions that we plotted. It is interesting as this probably means that the users have a consistent behavior: They click a lot because they query a lot, not only because they always query the same thing and click on all the links.

This assumption is confirmed by Figure 8 that shows the average number of clicks per query distribution over the users. Furthermore, we grouped users into intervals of average numbers of clicks per query in Figure 9 and, as we can see, 99% of users have between 0 and 10 clicks in average, per queries. Which indeeds confirms that the clicks behavior of users is quite stable. As a matter of information, the average of the *averages number of clicks per user* over the entire dataset is 2.02.

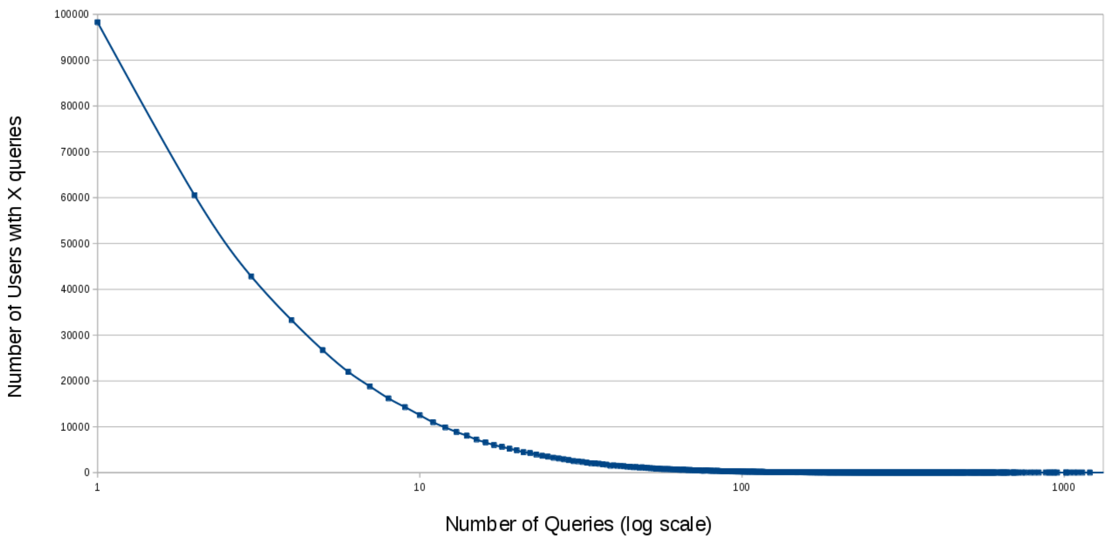


Figure 6: Queries Distribution: Users With A Given Number Of Queries

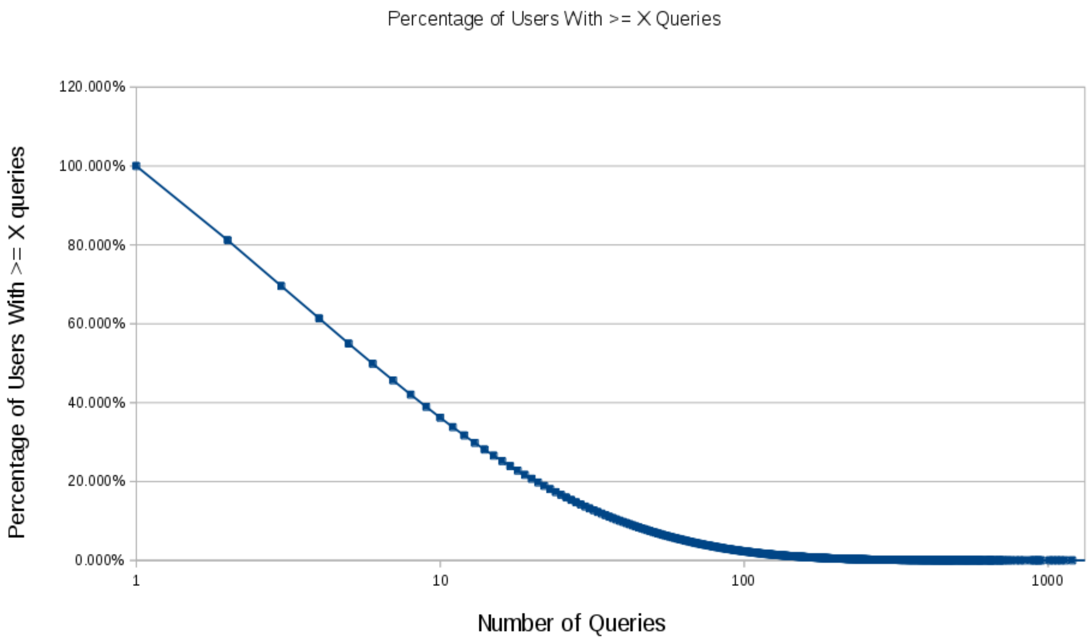


Figure 7: Queries Distribution: Percentage Of Users With More Than X Queries

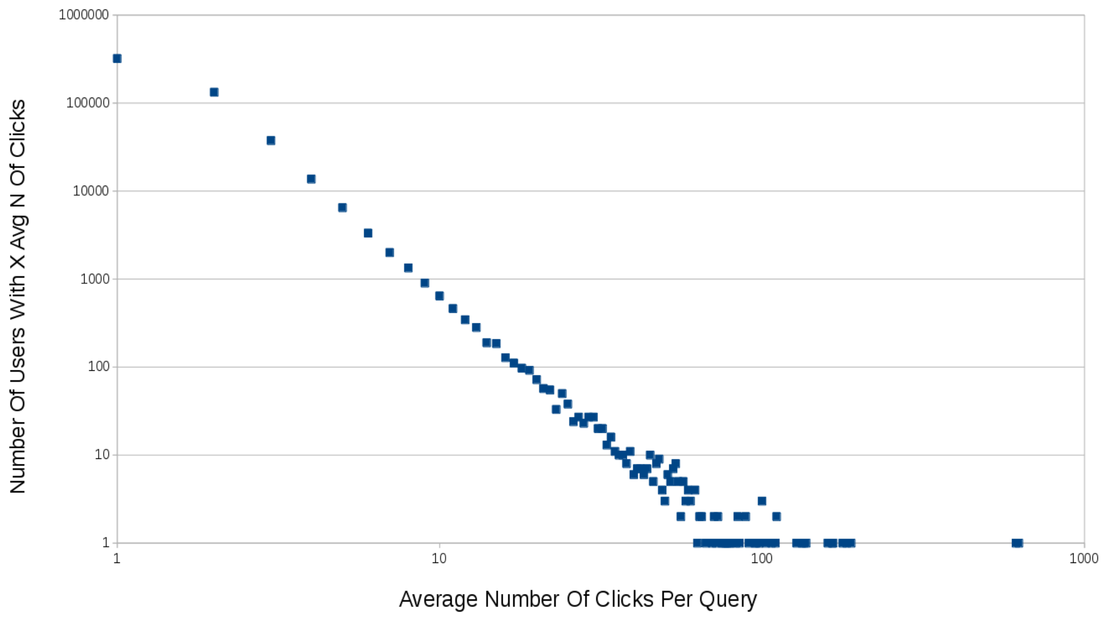


Figure 8: Users With A Given Average Number Of Clicks Per Query

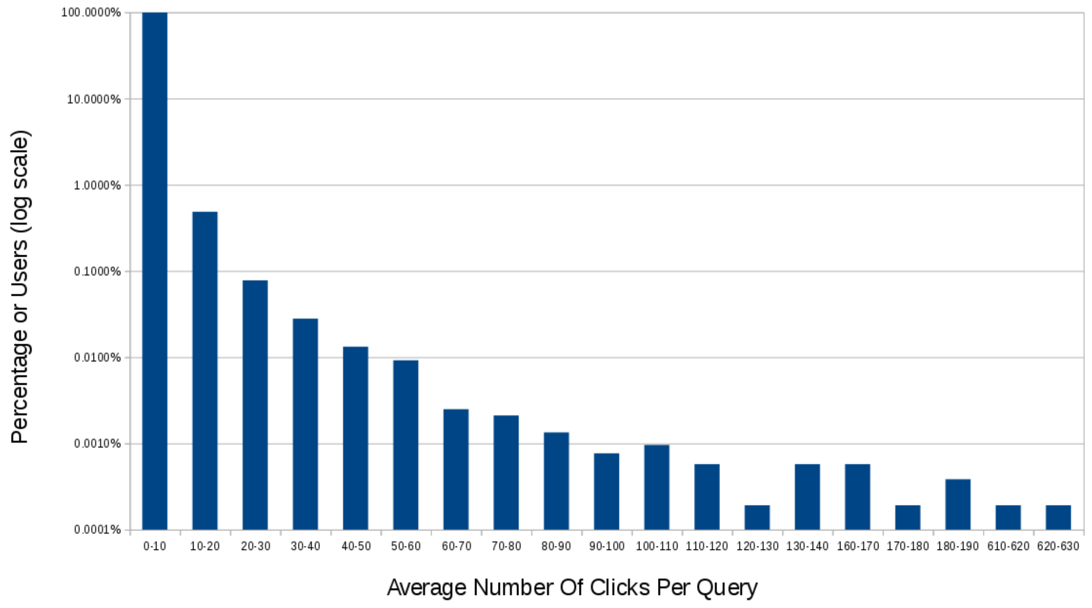


Figure 9: Percentage Of Users Within A Given Interval Of Average Number Of Clicks Per Query

11.3 Click Entropy Distribution

In order to make comparisons easier, we decided to compute the click entropy with the same formula as [20], as follows:

$$ClickEntropy(q) = \sum_{p \in \mathcal{P}(q)} -P(p|q)\log_2(P(p|q))$$

with $\mathcal{P}(q)$ the collection of web pages clicked for query q and $P(p|q)$ as follows:

$$P(p|q) = \frac{|Clicks(q, p, \bullet)|}{|Clicks(q, \bullet, \bullet)|}$$

If we simply look at the graph of the queries distribution in terms of click entropy by taking entropy values floored at 2 digits, it is a little bit hard to see anything, as shown in Figure 10.

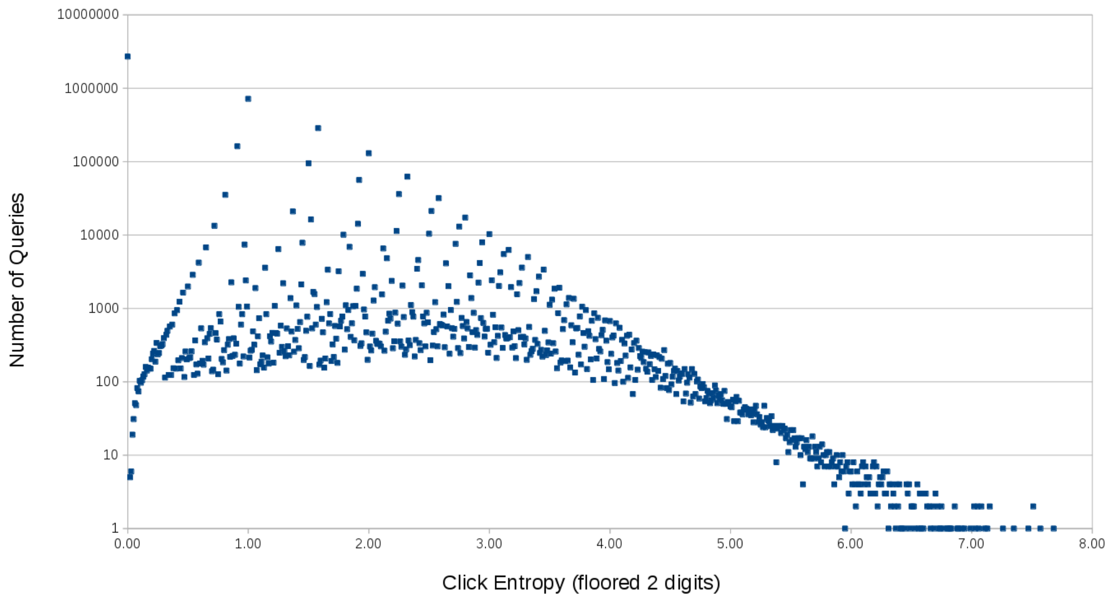


Figure 10: Click Entropy: Number Of Queries As A Function Of Click Entropy Score

However, we get a much better picture if we group them by interval of 1, for instance, as shown in Figure 11.

We also decided to plot the same graphs as the ones presented in [20]. That way, we believe we make it easier to compare both evaluations (and both dataset).

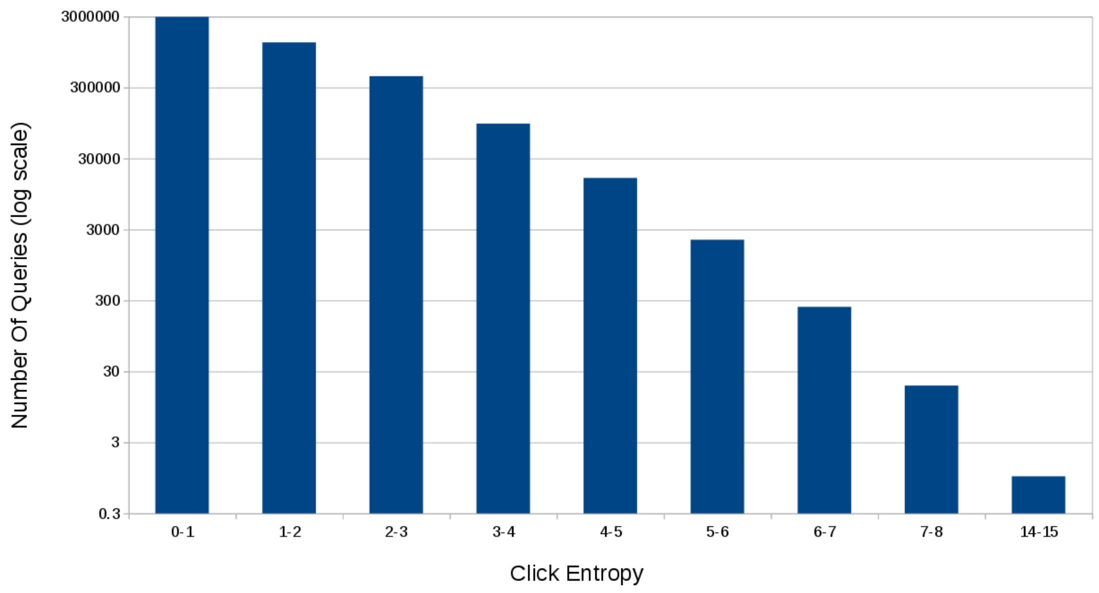


Figure 11: Click Entropy: Number Of Queries As Grouped By Click Entropy Score

Figure 12 shows the plot on our dataset on the left and the plot from [20] on the right.

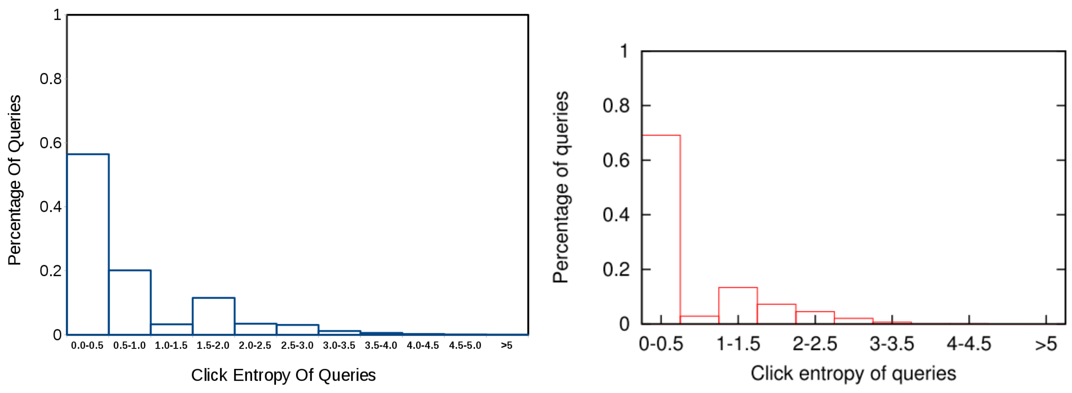


Figure 12: Click Entropy: Number Of Queries Grouped By Click Entropy Score. Left: Our Dataset. Right: from [20]

We can see here that distributions are quite different in some points. They both have a much higher percentage of queries with click entropy between 0 and 0.5, but inversed patterns for the next two intervals.

Starting at 2.0, though, both graph show a very similar pattern.

11.4 Clicked Items Ranks

It is generally believed that clicks are mainly made on the first page and especially on the top results.

We decided to plot the distribution of clicks for every *item rank* from the logs in order to see how it applies to our dataset.

Figure 13 shows that what is believed is fully true: First, the first page has an order of magnitude higher number of clicks, and the first item of the first page even higher. Also, on most pages (ranks from 1 to 10 and then $10^x + 1$ to 10^{x+1}, $\forall x > 0$), we see a much higher number of clicks for the first item of the page. It is often (but not always, as we will detail below) much higher than any other result on the same page, and then we see an exponential decrease until the end of the page. There is also a very sharp decrease after the first page, and Figure 14 shows that the 9^{th} item is the limit where the slope of the curve changes, thus meaning that the the decrease in clicks is very sharp between the 1^{st} and the 9^{th} item but then decreases slower.

Although one might think at first that on Figure 13 the patterns that we see show a constant decrease on every page and then the first item of the next page has more clicks than the last item of the previous page, it is in fact not always the case and quite often the last item of a page has much more clicks than the 9^{th} or 8^{th} on the same page. The item 25^{th} for instance, also has much more clicks than any other item on the second page, and the same goes for the 45^{th} item. We double checked our calculation in order to be sure it was not a bug on our side, but it seems the click behavior is just not constant over the 50 first pages.

The average item rank, weighted by the number of clicks, and over the entire set of clicks, of the dataset, is 6.89.

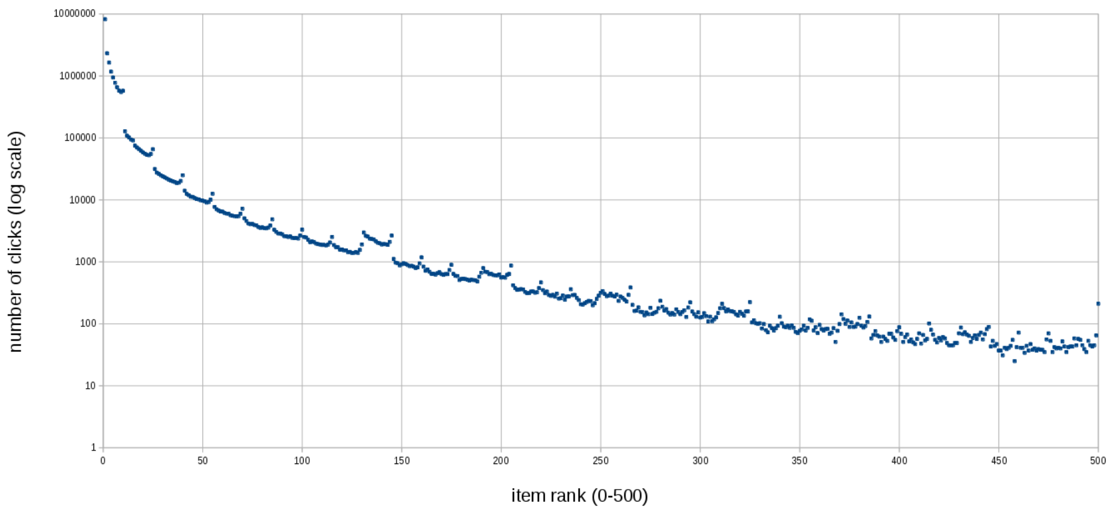


Figure 13: Clicked items: Distribution of Clicks per Item Rank

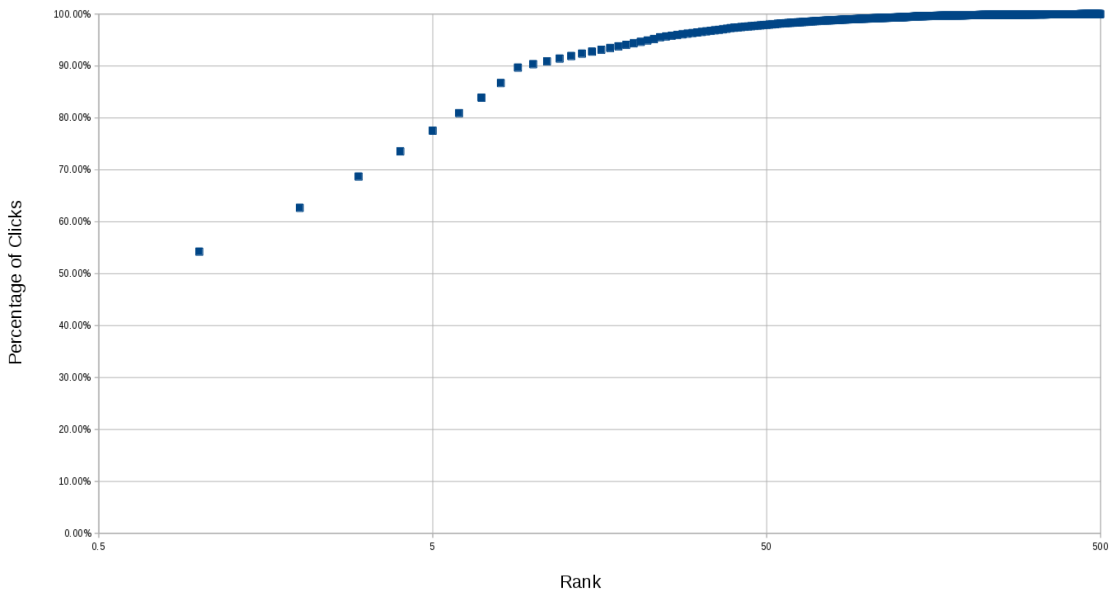


Figure 14: Clicked items: Percentage of clicks up to item rank X

12 AOL Logs Enrichment

12.1 Introduction

In this section, we are going to present how we placed ourselves in terms of use of the information contained specifically in the AOL Search Logs presented earlier. We are going to present the differences between what is needed in terms of information for our approach and what the dataset provides and present the technique we used to *enrich* the dataset with an approximation of the missing information.

12.2 Needed Information

While most re-ranking techniques (collaborative filtering, learning to rank) use the *current* ranking of the search engine and the *clickthrough* information, thus mainly working on the data from the dataset, our approach is based on using the *web graph* as a means of re-ranking. The needed information is thus scores to apply directly to the *web graph nodes.* Very few datasets out there provide enough information for a proper evaluation to be made. For instance, the Yandex dataset [51], while one of the most recent in search engines clickthrough datasets, is fully anonymized. Keywords, and more importantly, clicked URLs, are entirely replaced by unique identifiers. While one could try to perform statistical attacks on the dataset to try to deanonymize it, it is both against the terms of use of the dataset and also completely out of the scope of this thesis. Other datasets generally do not include URLs either. Sogou [52] dataset does include them but the queries are mainly in Chinese, and we would need a board of users speaking Chinese in order to evaluate queries from this dataset, which is this time simply out of reach for us. As a consequence, the AOL dataset is the one that provides the closest information to what we need, if we were to summarize, this is the list of the needed information:

- A unique way to identify users (id)
- A unique way to identify queries (id or query string)
- A unique way to identify clicked results/links (id or URL)

- Either search queries keywords (to be able to issue the query again) or the complete SERP ranking at the time of the logs.
  This is to allow an evaluation to be performed. If we have the keywords, we can do a user study-based evaluation. If we have the original complete ranking of every query, we can do an automated evaluation using a measure like NDCG [53], using the clickthrough data as a means of "relevancy" grading. This has been done in [54]. But these methods are inherently biased towards the set of top queries originally returned by the search engine [20] [39].

- URLs that have been clicked by the users
  This part is very important because if we do not have URLs, then we cannot map the clicked result to a node of the graph and we cannot provide personalized web graph nodes weights to the PageRank algorithm.

Now what does the AOL Search Logs provide us with, in terms of information?

- A unique way to identify users (user id)
- A unique way to identify queries (complete query string)
- The date and time of every action recorded in the logs
- A non-unique way to identify clicked results: the clicked URL **domain** and its *position* in the SERP.

The last bullet is where additional work is needed: The data in the logs is not entirely ready for us to be used. Only *domains* are provided. In addition to the domain, the original ranking (position in the SERP) of the clicked domain is provided, which allows us to differentiate, for the same query between two clicked domains, if they were not at the same position, as the search engine did not display twice the same URL (while it could have displayed several times the same domain (with some limit, depending on the search engine)).

As a consequence, we have an additional task here, before being able to use the dataset: we need to find a way to recover, as precisely as possible, the original URL, to be able to map its clicks to a web graph node.

12.3 Approximating Missing Information

What we decide to do in order to make up for the lack of unique and complete URLs is to map the clicks of a given (*domain*, *position*) pair from the 2006 logs, to a *similarly ranked* (*domain*, *position*) pair in the SERP that can be found by users *today*, a couple of years later (details about "*similarly ranked*" in chapter 14). We considered this option as the most likely to bring an equal semantic information as what could have been achieved at the time the logs were released. Indeed, while the web has evolved, some queries still are returning a lot of websites in common. If you were querying for an official US government form in 2006 for instance, if you query it today, it is very likely that the same page, of the same government website, will appear in the results.

As a consequence, we decide that we will only keep the information from the logs, that corresponds to queries that still have at least one domain in common in the 2014 SERP compared to the 2006 SERP.

We did this by re-issuing a pre-selected set of queries on the AOL search engine, and comparing the results of the first page returned. The re-querying system is explained in the next section.

12.4 AOL Re-querying System

The AOL search engine does not provide a direct API to issue queries and get results back. The only way is the "user way".

As a consequence, we simply write a script that queries the search engine like any *new* user would. We do not store cookies between sessions to insure that no kinds of personalization can be done. The fact that the consecutive queries have in general nothing to do with each other also probably prevents the search engine from performing personalization. Moreover, we also query the search engine from different IPs. Altogether, we randomly select a number of queries that we also issue by hand to check that the results were the same as the ones the scripts retrieved, and they were no evidences of any difference between what the script is able to return and what a human being would get in terms of results from the AOL Search Engine.

The system can be set up to crawl a given number of SERPs for each query (1 SERP = only the first page, 2 = first & second, etc.). For the current experiment, we proceed as follows:

We run it for "targeted" queries, the ones that were chosen to be part of the user study (see subsection 18.1.3). Those queries are re-issued until the 50^{th} SERP, as the logs included clicks up to the position 500, in other words, for the 50 first SERPs. We can do this because the number of queries to re-issue here is small.

13 Web Search models and tools

13.1 Web Crawl

In order to be able to run the PageRank, we need a web graph. Over time, there have been numerous interesting projects aiming at providing open web crawl datasets for research. Among them we can cite [55] which have put together an impressive amount of datasets, or the Stanford WebBase [56] that has been used in famous PageRank works in the past such as [2] and [6] but also many others. The most complete and recent one is probably [57] that is made available through the Amazon S3 services for convenience [58] but can also be obtained standalone.

Another web crawl worth noting is the Web Archives Association's one, which can be obtained upon request [59]. For completeness, we will not forget to also reference the *Lemur Project* [60] that has been already used in the IR field through different works in the past.

Although it seems at first that there are plenty of different web crawls available openly on the Internet, it is not as easy as it seems to use them.

Indeed, [57] for instance, represented in its 2012 version over 100TB of uncompressed data. Similarly, most of those datasets are either of unmanageable sizes if not run on some sort of distributed architecture, or are simply focused web crawls (like [55]) that will not necessarily include the web pages that are referenced from the logs. Also, except one or two maintained projects, the web crawls are also relatively old. Moreover, some of them only include the graph and not the actual data of the web pages, and thus we would not be able to run an IR request against them.

As a consequence of these drawbacks, we decide to give a try to write our own web crawler and create our own focused web crawl around a set of links extracted from a set of queries that we want to study in the context of the experimentation and evaluation of our research work. Having experience with the *Scrapy* [61] framework to build web crawlers, it is quick enough to build our own one.

The advantage of building our own one is also that we will be able to give it the behavior we want. For instance, we might prefer crawling many different websites compared to crawling all pages of a given website, as we are interested in PageRank computation afterwards and as the data we have from the logs is per domain.

Our web crawler behaves and is configured as follows:

- It starts from a URLs seed list (seed generation will be described later).
- It crawls up to a depth of 5. "Depth" represents the number of consecutive links we followed since the seed URL + 1 ($A \rightarrow B \rightarrow C$, C is at depth 3).
- It follows only links that are less than 255 characters long.
- It does not crawl a list of predefined patterns (cgi, images, ...) including the most common authentication systems like Google/Facebook/Microsoft authentications systems.
- It processes links in the order of the HTML code.
- It gives priority to a "broad" crawl (many different websites/domains), compared to crawling all pages of a website:
  - It follows at most 3 links to the same domain, per page.
  - It follows at most 7 links to domains different from current one, per page.
- It does not follow redirections (301, 302).
- It obeys the `robots.txt` directives.
- It waits at maximum 60 seconds to download a page.
- It may retry up to twice to download a given web page.

- It processes at most 500 request in parallel.
- It does not store nor sends back cookies.
- It performs at most 8 concurrent queries to the same domain.
- It waits 1 second between queries to the same domain.

Such settings/rules are a combination of what we want to achieve and "best practices" for fast broad web crawls we have found when searching.

We run 3 instances of the web crawler in parallel, with different parts of the URLs seed list, in order to take advantages of all the CPUs of the server.

The URLs seed was generated by taking the 500 first links returned by the 2014 AOL Search Engine for the queries that were selected.

The selected queries are defined in subsection 18.1.3.

13.2 Information Retrieval models and tools

13.2.1 Introduction

In this section we are going to present the different tools and models we used in terms of Information Retrieval. Although this thesis is not focused on IR techniques but rather on a non-IR technique that is used in addition to IR techniques in web search to improve the results, we still need to use IR tools as part of our experimentation if we want to rebuild search-engine-like system. This section is going to first present what kind of "search engine" we are building. Then, it will present the tools that we used (ElasticSearch) and how we configured it. Next, we will explain the details of how we indexed the previously described web crawl in the search engine, especially the pre-indexing processing. We will finally briefly explain how we query the search engine in order to give all the necessary details for our work to be reproducible.

13.2.2 Web Search Engine

In order to reproduce the real-world process in which our proposed PageRank personalization technique would be used we needed to somehow build a "web search engine". Needless to say that one does not simply build a concurrent to Google in a matter of months. So, our *search engine* is not anywhere near Google in terms of both quality of results (which also mostly comes from the web crawl quality) and completeness and speed. The goal of this *search engine* is to allow us to make an information retrieval query on the web crawl we created. Once this query returns results, we want to then apply both standard PageRank and CPPR scores to the set of results and display the rankings combining scoring from both IR and PageRank, the same way it has been done in [2] and [6].

A simplified view of the setup we want to achieve is presented in Figure 15.

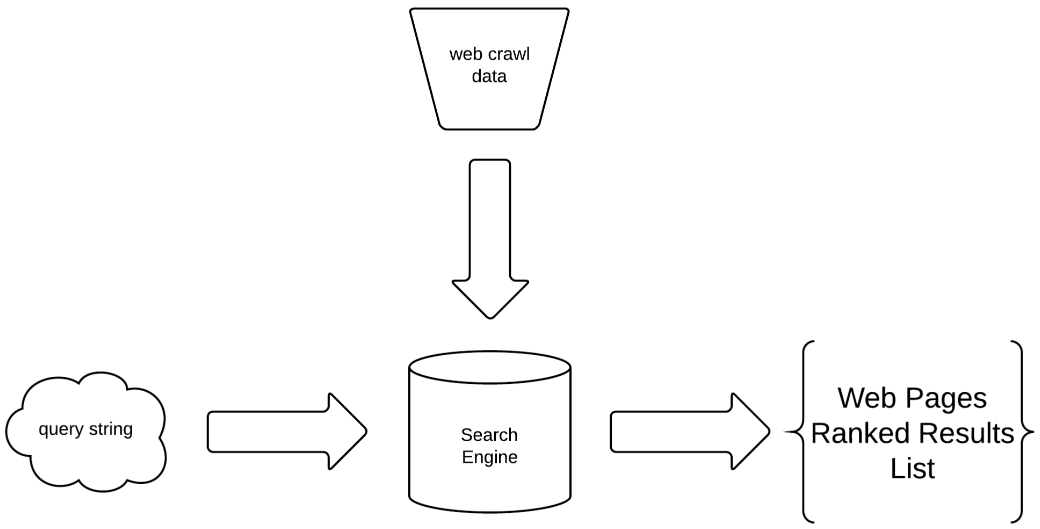


Figure 15: Simplified Web Search Engine Setup

13.2.2.1 ElasticSearch & BM25

The criteria for the tool to be chosen as our *search engine* were the following:

- It has to run as a standalone application
- It has to be queried easily
- It has to provide an easy indexing API
- It has to provide the state-of-the-art in terms of IR techniques, more specifically: BM25.

In terms of open source software for information retrieval, several options came to us.

First, the IR Framework Terrier [62] seemed to be an interesting candidate. But having previous experience with it, we knew it is not straightforward to add new features to Terrier, especially if they do not only rely on standard indexing techniques. Also, we wanted a solution that could be queried easily through an API, well documented and if possible widely used so that it has proven to be stable.

Then of course came to mind the IR library Lucene [63] from the Apache Foundation. Lucene is an interesting candidate but it is a *library*, not a full-blown search engine. We would have had to develop again some parts of the search engine. Moreover, Lucene has no built-in scalability features and we did not know yet if indexing between half a million and a million documents would require much resources or not.

Built based on Lucene, the two main projects that we had to choose from in the end were Apache Solr [64] and ElasticSearch [65]. Solr had the advantage of being integrated with the Nutch [66] project that basically is a web search engine, including crawling and indexing.

However, Nutch has in our opinion one of the worst documentation one could find. It also appeared that Solr did not scale very well as of the reviews we could find. Moreover, ElasticSearch offered the option to use the Lucene's BM25 implementation easily and provided an API based on JSON, which fulfilled our criteria.

Having previous experiences with ElasticSearch, it appeared as the best option as the documentation is much more complete. It also is probably the currently most recognized and stable open source deploy-your-own-one IR search engine as of this writing.

The details of the indexing configuration of ElasticSearch will be given in the next section.

13.2.2.2 Indexing: Pre-processing and indexing settings

First of all, we configure ElasticSearch as follows, to index the web crawl:

- 4 shards, in case we need to scale to multiple servers
- 0 replicas

The exact configuration follows:

```
http.max_content_length: 500mb
index.store.compress.stored: true
index.store.compress.tv: true
index.cache.field.max_size: 50000
index.cache.field.expire: 10m
```

We then create a *mapping* to store the data about each web page, as follows:

```
doc_type: {
  "_all" : {"enabled" : "false"},
  "_source" : {"enabled" : "false"},
  "_id": {"store": "true"},
  'properties': {
      'url': {
        'type': 'string',
        "store": "true",
        'index': 'not_analyzed'
      },
      'title': {
        'type': 'string',
        'similarity': 'BM25',
        "store": "true"
      },
      'content': {
        'type': 'string',
        'similarity': 'BM25',
        "store": "true"
      },
      'description': {
        'type': 'string',
        'similarity': 'BM25',
        "store": "true"
      }
  }
}
```

This mapping is composed of the following main components:

- It indexes the title, the content of the web page and its description using BM25. Indeed, this is what we are susceptible to use when querying the search engine. We did not know yet if we were going to only use the content of the page or not. This would depend on how the results would be.
- It does not store the `_all` field. Indeed, this would replicate all fields combined into one main one, and double the index size, which is totally useless for us.
- It also stores the URL string, that will be displayed when showing the results and also to allow visiting the link if the title and the descriptions are not enough to guess correctly the content of the web page.
- It stores the `_id` field “as a field”, this `_id` field is going to be assigned directly by us (rather than letting ES choose a unique id) to the unique integer that has been assigned to every URL during the web crawl. We thus want to retrieve it as we will need the URL unique identifier in order to apply the personalization. To retrieve it, it need to be stored “as a field”.

Once ElasticSearch is set up, the indexing process goes as follows:

- Load the entire web graph as crawled.
- For every unique link in the graph, try to load the GZipped downloaded web page from the disk.
- If no such file exists, then this web page was on the border of the crawl (and thus not crawled) or it had not been downloaded for other reasons (the server did not respond, refused crawlers, sent a wrong HTTP code… or it was excluded by the robots.txt file).
- If such file exists, load its content, *process it* and store it in memory up to a given batch size.
- When a batch is ready to be committed, send the batch to ElasticSearch.

The *process* we apply to the content of the web page is the following:

- First, we try to guess its encoding using the library *BeautifulSoup4* [67], which is a renowned library for HTML parsing (including faults-robustness/ invalid HTML robustness).
- If BeautifulSoup4 was not able to guess the encoding, we try the most common encodings (utf–8, latin1, latin9, windows-latin1).
- If still no proper encodings are found, we use the library *Chardet* [68] that is specialized in guessing encodings from text data.
- In case nothing is found we force conversion into UTF8 and remove the incompatible characters.
- Once the encoding has been guessed, we process the document HTML data with BeautifulSoup4 as follows:
  - We extract the <`title`> tag if any, and use this as the `title` field (cf. ES Mapping (subsubsection 13.2.2.2)) of the web page.
  - If no <`title`> tags are found, we look for the first <`h1`> tag in the HTML and use this as the `title` field.
  - In case there also are no <`h1`> tags, we assign "Untitled" as the document `title`.
  - We look for the <`meta`> tag with `name="description"` and store it in the `description` field if found.
  - We remove comments.
  - We remove CSS code.
  - We remove Javascript code.
  - We strip all HTML Tags, convert everything to text and store it in the `content` field.

The indexing process turns out to be quite slow. Running it on 1 CPU on a server with not very good I/O performances ran at a pace of 1.6 indexed pages per second. On a server with better I/O performances, we could go up to 4.5 pages/second.

As a consequence, in order to index the final web crawl of a million web pages in less than 30 hours, we ran several workers on several servers, all committing to a central ElasticSearch server as shown in Figure 16.

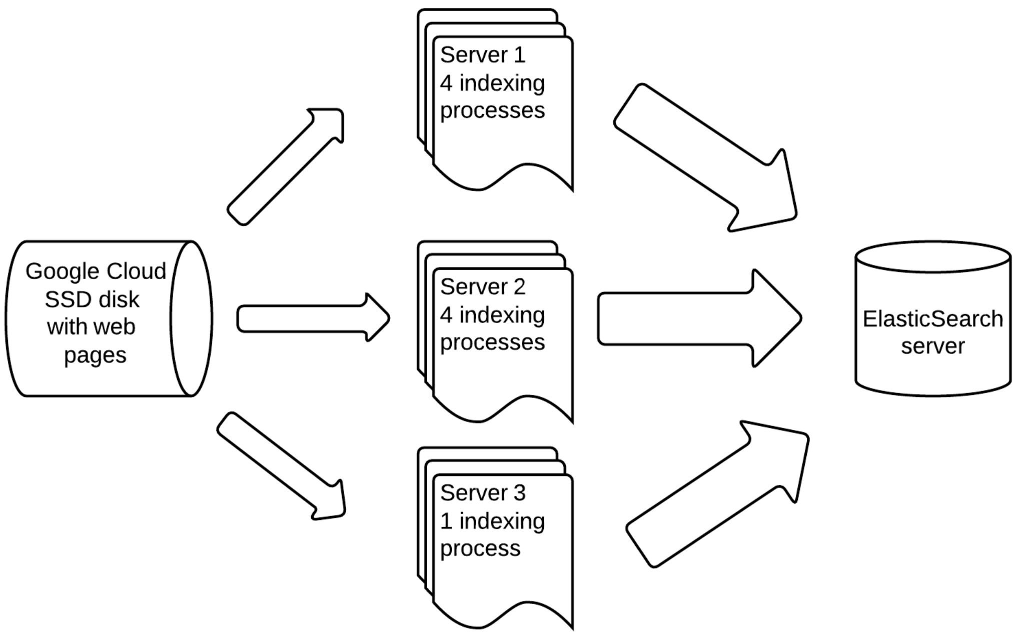

Google Cloud SSD disk with web pages
Server 1 4 indexing processes
Server 2 4 indexing processes
Server 3 1 indexing process
ElasticSearch server

Figure 16: Indexing Architecture

13.2.2.3 Querying

ElasticSearch provides multiple ways to *query* its indexes. Depending on the way indexes are queried, results might change.

For instance, it is possible to query without specifying a field, in this case ElasticSearch will give more importance to fields that are shorter.

In order for our work to be reproducible, we provide here the syntax, using the *ElasticSearch Query DSL*, we used in order to make a query to ElasticSearch:

```
{
    'query': {
        'match': {
            'content': "the query string"
        }
    }
}
```

In addition to that, we asked ElasticSearch to return the fields `_id`, `_score`, `url`, `title` and `description`. But the syntax for this is specific to the Python driver we used and we will thus not provide it here.

The `_score` field is the matching score of each document against the query string, as far as we know it is directly coming from the Lucene engine used under the hood by ElasticSearch.

14 2006-to–2014 URLs Mapping

The analysis of AOL 2006 logs (chapter 11) using the technique presented in section chapter 7 brings us a set of (*user*, *query*, *domain*, *score*) personalization tuples.

However, three issues arise with this:

1. The score is global to a *domain*, which might not be precise enough depending on the query.
2. Domains in the logs date from 2006 and the web graph we are going to generate by crawling the web will be from 2014.

3. The CPPR vectors presented in section 9.4 are using URLs from the web graph we actually work on. The one we crawled using the crawler defined in section 13.1. Those are full URLs, not just domains.

Thus, we need to create a mapping $URL\ in\ 2014 \rightarrow domain\ from\ 2006\ logs$.

One could think we could simply reduce the 2014 URLs to their domain part and the mapping would be over. However, this rises the following issue: A URL A, belonging to domain B, could be assigned some clicks (and thus weights) for query $q1$ and $q2$ that both reference domain B in the logs while in fact those two queries are quite different and the pages, even from the same domain, that appeared in the AOL SERP in 2006 were probably different.

As a consequence, we need to build a more specific mapping, namely, we need to build a mapping $(query,\ URL2014) \rightarrow (query,\ domain2006)$.

But how do we know if URL u from the 2014 web crawl should belong to query $q1$ or $q2$ for instance?

What we decide to do it to use the AOL 2014 SERPs and AOL 2006 SERPs as a way to generate this mapping.

What we do is the following, for every targeted query:

- We take every URL clicked from the logs.
- We load all the URLs that were returned in the 2014 SERPs when re-querying the AOL Search Engine (see section 12.4).
- We walk through the list of URLs in the order they appeared in the 2006 SERP:
  - For each one, we walk through the URLs from the 2014 SERP of this same query, **also in order of the SERP**.
  - If domains are the same, we create a mapping entry $(query, URL\ id) \rightarrow (query, domain\ id)$ and we remove both entries (domain (2006), URL (2014)) from the lists so that they cannot be mapped again.

The algorithm can be described in pseudo-code as follows:

```
foreach query in set_of_targeted_queries do
    foreach domain in clicked_urls_from_logs_sorted(query) do
        foreach url in 2014_aol_serps_sorted(query) do
            if extract_domain(url) == domain then
                $mapping[(query, url)] \longleftarrow domain$
                $remove\_first\_from\_logs(query, domain)$
                $remove\_from\_2014_serps(query, url)$
            end
        end
    end
end
```

Where $remove\_first\_from\_logs(query, domain)$ will only remove the first (in the SERP order) entry of *domain* for the query *query* and $clicked\_urls\_from\_logs\_sorted(query)$ and *2014_aol_serps_sorted*$(query)$ respectively return the clicked domains and the URLs *in the order of the SERP* (top-ranked item first).

Then, once this mapping is in place, we need to run the formula from chapter 8: $score(u, q, p)$. We use it as follows: $score(u, q, mapping[(q, p)])$ (recall: q is the query, u the user and p the web page (url))

If there are no mappings, the call to $mapping[(q, p)]$ returns a special value. The function $score()$ then returns the standard score (uniform PageRank score, equal to $\frac{1}{|graph|}$) if this special value is passed as a parameter, instead of returning the personalized score.

15 PageRank Computation

15.1 Choice Of The Library

When it comes to actually implement PageRank computation, it seems obvious to us that this is an already widely solved problem and we should not create our own implementation. Both because this would be wasted time and also because our implementation would likely be less tested, reliable and even potentially optimized than long-proven libraries.

Among the library we have tried: JUNG (Java), Graph-Tool (Python, C++), iGraph (Python). But we have also studied many others. As often with libraries, criteria of choices were: simplicity of use and of integration, completeness and clearness of documentation, presence of tutorials, community support and performances. The required technical features were: PageRank computation, exposure of the PageRank *personalization vector* and input/output in widely used graph file formats (such as GraphML).

Java-based libraries quickly showed that they did not fulfill the first goal. A quick test at JUNG took us to the creation of half a dozen classes in order to read an XML file and transform it into a graph. We could not even imagine how complicated it would be to integrate with the entire project.

Graph-Tool, which seems to be currently maintained by a single developer, researcher in the German University of Bremen, turned out to perfectly fulfill our needs. In terms of performances: it is written in C++ and provides very interesting performances, it is built with OpenMP [69] support, which allows it to leverage the power of all available CPUs on the host system. In terms of ease of use: the exposed API is very *pythonic*, that is to say that it respects the main Python recommendations [70], such as *there should be only one obvious way of doing things* [71]. Moreover, the rest of our project being in Python, the integration of Graph-Tool was literally straightforward. In terms of features: It has built-in input/output in the standard GraphML format, with input/output being directly integrated in the library and done at the C++ level, making it hugely faster than loading the graph from Java or Python (and this counts, when the graph has 17M nodes and 80M edges). The community support was also very good as the bug tracker is very active and the main developer timely to answer. We, by the way, thank Tiago de Paula Peixoto, the developer of the library, for his incredible work and also for his responsiveness on the bugtracker.

When ran a couple of performances tests on Graph-Tool before settling with it. On a i7–4600U machine, we were able to compute PageRank in about one second on graphs up to 500k nodes and 5M edges. At the time of performances tests, we still had in mind the possibility of a real-time application and thus wanted to keep the computation time under one second. Loading such a graph took about 15 seconds.

15.2 From Web Graph to CPPR Pre-Computed Scores

After running a couple of web crawls, we noticed that the number of edges in a web graph was way too high for us to keep the CPPR "application example" a real-time application. Moreover, as the user study and the evaluation are going to be run on a previously selected set of targeted queries, the need for a real-time *query* → *results* application is inexistent. We thus decide to pre-compute the CPPR scores, using a targeted set of (query, user) pairs.

From the output of the web crawler (section 13.1) to the pre-computed CPPR scores, the system can be described by the diagrams shown in Figure 17 and Figure 18.

Figure 17 shows the post-processing that is necessary on the output of the multiple crawlers that were run in parallel until the generation of the final, merged graph that is usable by the Graph-Tool library.

Figure 18 shows the process of generating the CPPR scores. The input information is the previously computed sets of top-similar users, clicks profiles, personalization scores and the web graph. And the output is the CPPR scores.

However, the final usage of such scores is going to be applied only on a subset of the URLs of our web graph. Indeed, the CPPR score is combined with the results of the IR engine (section 13.2). Thus, instead of storing millions of useless scores in database, what we do is that we only store the ones that are mapped to the IR engine output. This is explained in more details in section 15.3.

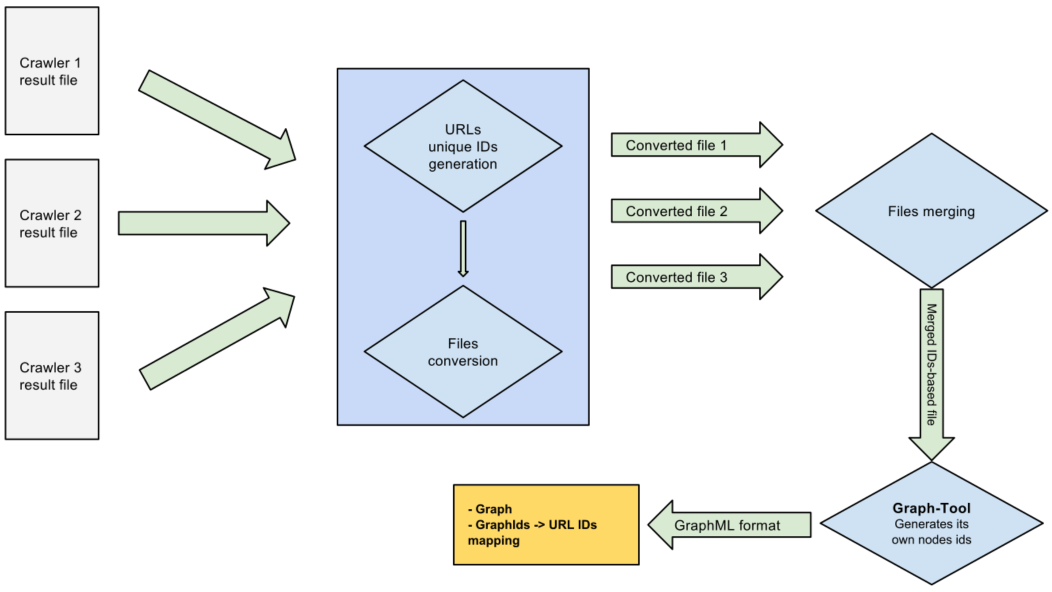


Figure 17: Process Flow from Web Crawler Output to Graph-Tool GraphML file

15.3 Final Rankings CPPR Scores

As previously stated (section 15.2), storing directly the output of the CPPR computation would store millions CPPR values for millions of URLs (all URLs of the web graph) while we in fact only need to store the ones that will actually be used.

The ones that will actually be used are URLs that are returned by the IR engine (section 13.2) for the set of targeted queries of the evaluation (subsection 18.1.3). As a consequence, we decide to directly generate the final ranking, and to compute the CPPR values only as part of the process of generating the final rankings.

In order to do that, the system goes through the following steps:

1. It reads a list of (query_id, query_string, user_id) triples.
2. It queries ElasticSearch using the query_string and asks the 1,000 top results to be returned.

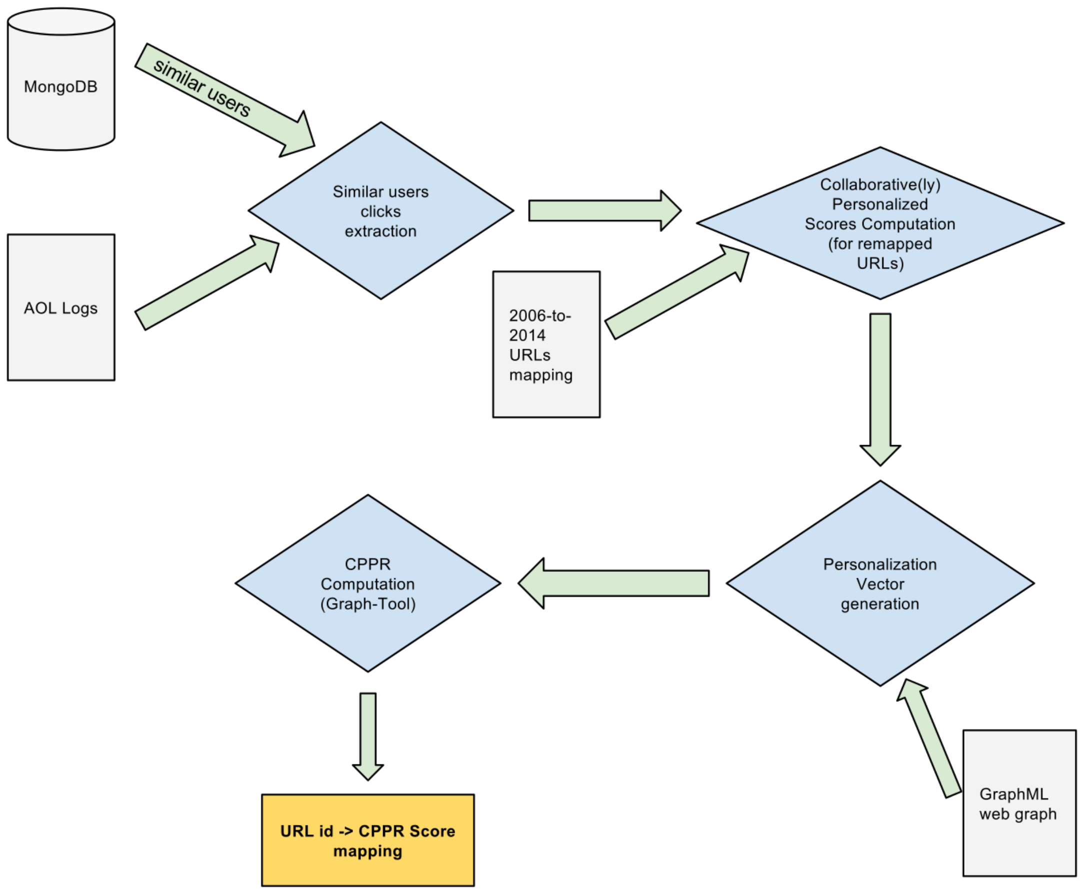

MongoDB
similar users
AOL Logs
Similar users clicks extraction
Collaborative(ly) Personalized Scores Computation (for remapped URLs)
2006-to-2014 URLs mapping
Personalization Vector generation
CPPR Computation (Graph-Tool)
GraphML web graph
URL id -> CPPR Score mapping

Figure 18: Process Flow from GraphML to CPPR Scores

3. It loads the CPPR computation engine and initializes it with the necessary file paths and a connection to the DB (see Figure 17 and Figure 18 for the required input of the CPPR computation engine).
4. It asks the CPPR computation engine to first compute once the normal, non-personalized PageRank (will be used for the *baseline* rankings).
5. Then, for each triple:

    a) It asks the CPPR engine to compute the CPPR vector on the entire graph.
    b) It keeps only the scores corresponding to the URLs IDs contained in the results returned by ElasticSearch.

c) It stores in the DB the corresponding triple and the top 1,000 results together with their IR score and CPPR score.

This process is illustrated in Figure 19.

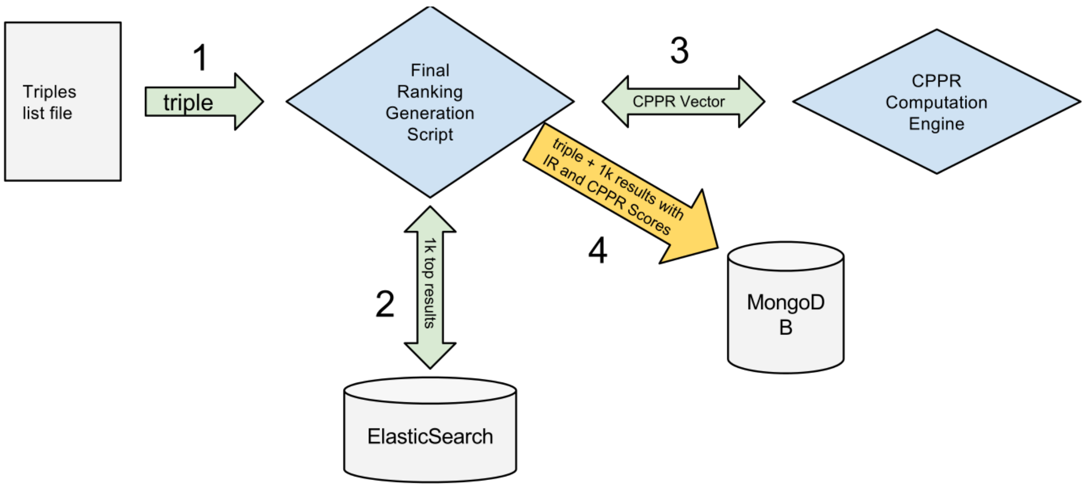


Figure 19: Final Rankings Generation Process

You have probably noticed that no composite scores have been generated yet. This is because composite scores are computed online. Computing composite scores online allow us to experiment with different rank merging algorithms. This is discussed in chapter 16.

16 Rank Merging & Post-Processing

Rank merging is a research domain by itself. As a consequence, it is not feasible for us to review all rank merging algorithms out there to see which one would perform best. The main reason for that is that if we really want to compare them we would basically need to run a user study for each rank merging technique to see how well they perform in the specific case of our application.

As a consequence, we decide to experiment with only two rank merging techniques: Multiplication of scores and Borda Count.

While it may not seem like a "real" technique, multiplication of scores is what is done in [6] for instance but also in the web search engine Nutch [66]. So we decide to give it a try, especially considering the straightforwardness of implementing it.

On the other hand, Borda Count [72], also called Borda Voting Scheme, is a technique that is commonly attributed to the French mathematician Borda for his works during the 18th century. It is used in some parts of the world as a way to elect candidates during political elections. The details of the algorithm are not discussed here but can be found online [73] or in the literature [74]. The reason we test Borda Count is because it is the rank merging technique that was used in [20] of which we already reuse the collaborative click-based scoring approach and they had good results with it. Considering the fact that we use only a slight modification of their scoring approach and their results with Borda, we consider it is an interesting option to experiment with it.

As we do not have the possibility to run two user studies, we simply try to differentiate them by ourselves and ask the opinion of two other people on cases where we can not decide easily.

It turns out that Borda Count and the multiplication of scores actually return extremely similar results and even often the same results, at least for the top 10 results, that are going to be displayed to the users. In some cases, the multiplication of scores seems to return more relevant results, thus we decide to keep it, as it is also the technique used in [6] and in Nutch, precisely to merge the PageRank scores and the IR scores.

Thus, once the results are retrieved from the database, the composite score is computed by multiplying the IR score and the PageRank score. The PageRank score being the "normal" PageRank score for the *baseline* ranking and CPPR score for our personalized ranking. Once composite scores are assigned, items are sorted in descending order of composite score.

Once all items are sorted, we display them with a limit of 2 items per domain in the final rankings. This limit has several reasons: First, without setting a limit, the IR engine would generally have returned a very high number of results from the same domain, and the PageRank of the different pages on the domain is often quite similar. This is because webmasters use strong self-linking strategies to mutually improve the PageRank of the web pages of their websites. This is a problem Google dealt with a couple of years ago but Google only knows how they did it. Second, a limit of results per domain set to 2 (or 3) is also what Google used to do until just recently (they seemed to have changed it to a variable limit) and is what other modern web search engines do. As a consequence, we consider using such a limit actually makes our rankings more similar to real-world search engines and as such, our evaluation closer to real-world results that could be obtained in the industry.

17 Overall Picture Of The Entire System

As the overall system is quite complex, we provide here a global picture of the CPPR personalization system before diving into the evaluation of the system. Figure 20 shows an example of how the system could be implemented for instance when integrated to an existing search engine, which would already have an IR engine and a web graph in place and which would have the same URLs in its logs, IR and web graph engines.

On the other hand, Figure 21 shows the experimental setup that we used in the current experiment. As one can see, the required number of parts in the systems, and specifically of processes, is much higher. This comes from the fact that the logs are not recent and coming from a web graph engine and web search engine that we do not have control over and thus we need additional steps to rebuild the web graph and remap the URLs. The additional processes of selecting the targeted queries and users for the experiment/evaluation are also not present in the case of the industry example.

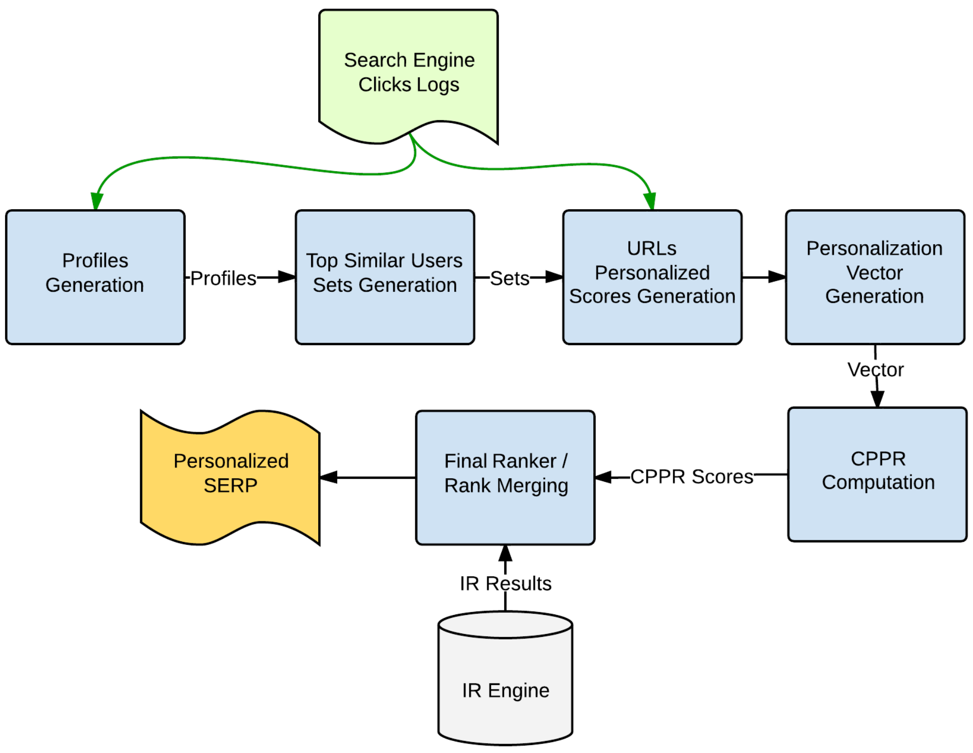

Search Engine Clicks Logs
Profiles Generation
Profiles
Top Similar Users Sets Generation
Sets
URLs Personalized Scores Generation
Personalization Vector Generation
Vector
CPPR Computation
CPPR Scores
Final Ranker / Rank Merging
Personalized SERP
IR Results
IR Engine

Figure 20: Example of Overall System In an Industry System

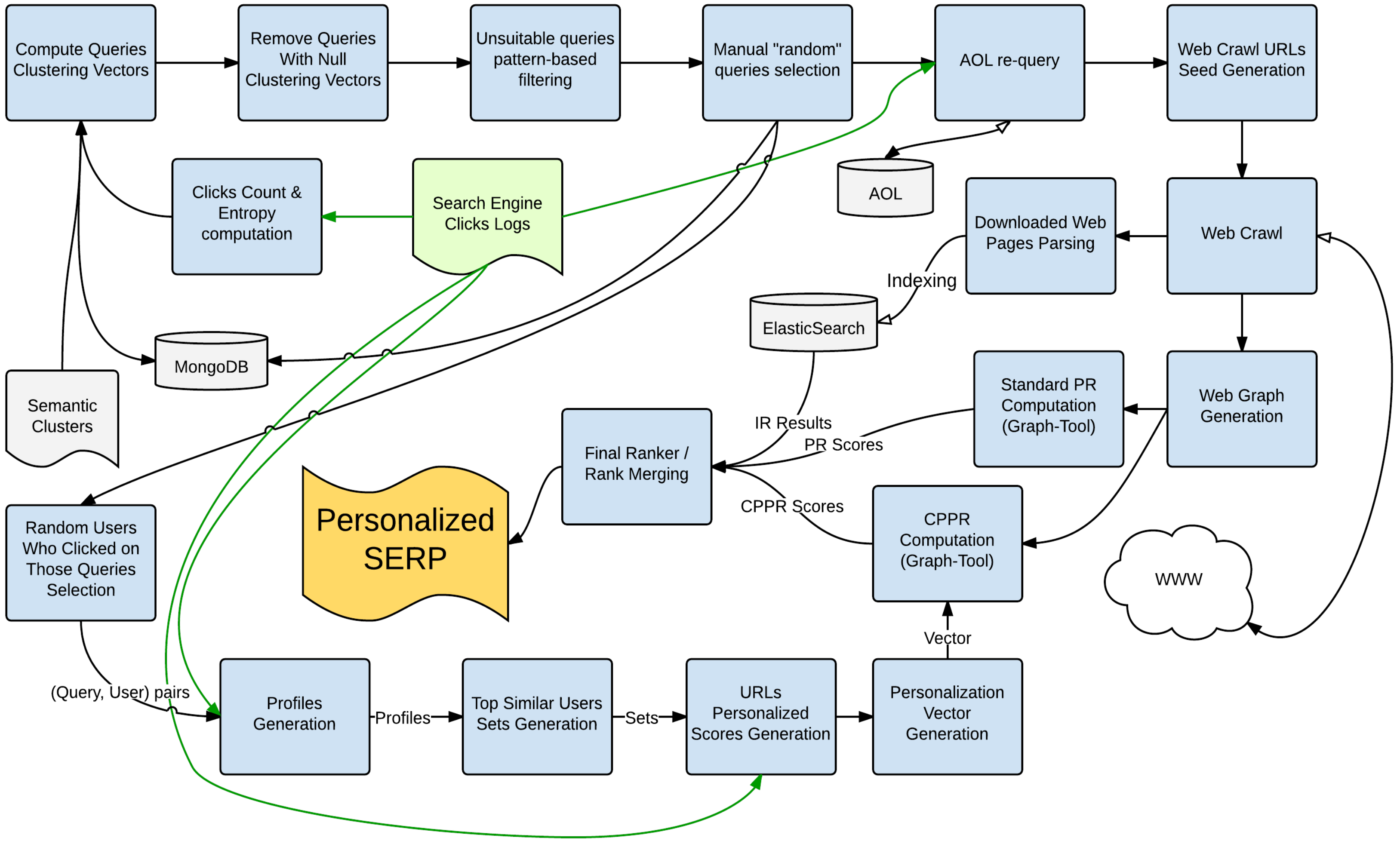

Compute Queries Clustering Vectors
Remove Queries With Null Clustering Vectors
Unsuitable queries pattern-based filtering
Manual "random" queries selection
AOL re-query
Web Crawl URLs Seed Generation
Clicks Count & Entropy computation
Search Engine Clicks Logs
AOL
Downloaded Web Pages Parsing
Web Crawl
Indexing
ElasticSearch
MongoDB
Semantic Clusters
Standard PR Computation (Graph-Tool)
Web Graph Generation
IR Results
PR Scores
Final Ranker / Rank Merging
CPPR Scores
CPPR Computation (Graph-Tool)
Personalized SERP
Random Users Who Clicked on Those Queries Selection
WWW
Vector
(Query, User) pairs
Profiles Generation
Profiles
Top Similar Users Sets Generation
Sets
URLs Personalized Scores Generation
Personalization Vector Generation

Figure 21: Complete Experimental Setup View

18 Evaluation

In this section, we are going to present how we evaluated the performance of the experimental setup described in Part IV up to now.

18.1 Methodology

18.1.1 User Study Protocol

In order to make our work comparable, we decide to conduct the evaluation following a methodology in most points similar to the one conducted in the closest previous work to what we are doing, that is to say [6].

In [6], the authors choose 5 volunteers who are presented with 10 different queries and their results using both the standard PageRank and the Topic-Sensitive PageRank. Volunteers are asked two things:

1. Select all results (links) that they think are relevant to the given query.
2. Give an overall preference for one of the two presented rankings.

Point 2. gives them the overall appreciation on whether their algorithm at least gives the impression to the user that results have been improved (which is an important point, as user satisfaction is the main goal of most systems).

Point 1. allows them to compute a *precision* measure. [6] defines the *precision* $p(q)$ for a query q as follows:

$$p(q) = \frac{|relevant\_results(q)|}{10}$$

with $relevant\_results(q)$ the set of results that are deemed *relevant.* A result is deemed *relevant* in [6] if at least 3 of the 5 users marked it as relevant. The constant 10 is the total number of results in the displayed ranking.

The authors say that volunteers "were not told anything about how either of the ranking was generated".

However, the authors do not specify:

- Whether the same ranking (left/right, or first/second displayed) was always the non-personalized and the personalized one.
  This is important because the human brain, even on as few as 10 queries, can definitely develop a preference over the first or second ranking if for instance in the 3 first queries the second ranking was much better. The user will start paying more attention to some details in one of the ranking and not in the other one, for instance.
- It is not entirely clear whether the users were only presented with raw "URLs" or with a snippet of the content, the title of the page or not. But the text seems to say that they were only presented with raw URLs.
- Whether the users had the possibility to actually browse the web page corresponding to the URL in order to get a glance at what its actual content was or not. We believe users were simply supposed to *judge URLs* which in our opinion does not make enough sense as a URL could well often not be quite representative of the actual content of the web page.

While the user study protocol had some imprecisions, the limits of their evaluation can also be found in the way they present results. Indeed, in the case of the "preference" of one ranking compared to the other one for instance, they do not publish the actual figures, but a rather fuzzy measure as "preferred by majority" for each query, as shown in Figure 22. "Majority" not even being defined, but we guess that, for 5 volunteers, it means 3 volunteers, so 60%. However, one query is marked as having a preference "neither" for one ranking nor the other (see Figure 22). However, 50% is not achievable with 5 users, which means some information is missing in order to fully know how their "preferred by majority" measure is calculated.

In our evaluation, we follow a similar protocol. We find 10 volunteers, and select 5 (queries, user) pairs. The latter pairs are not completely randomly selected, the details about how queries are selected are presented in subsection 18.1.3.

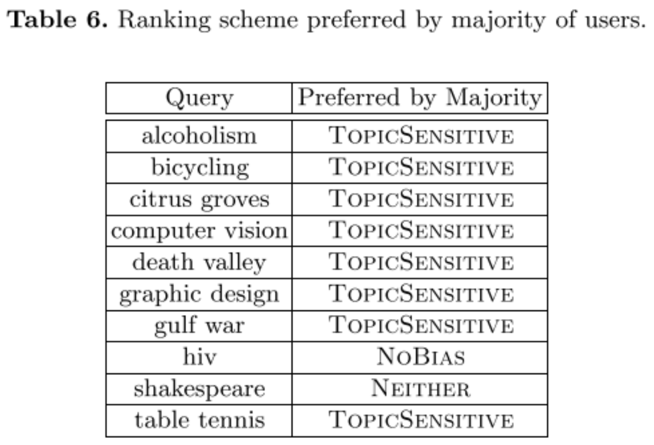


Table 6. Ranking scheme preferred by majority of users.

Query	Preferred by Majority
alcoholism	TopicSensitive
bicycling	TopicSensitive
citrus groves	TopicSensitive
computer vision	TopicSensitive
death valley	TopicSensitive
graphic design	TopicSensitive
gulf war	TopicSensitive
hiv	NoBias
shakespeare	Neither
table tennis	TopicSensitive

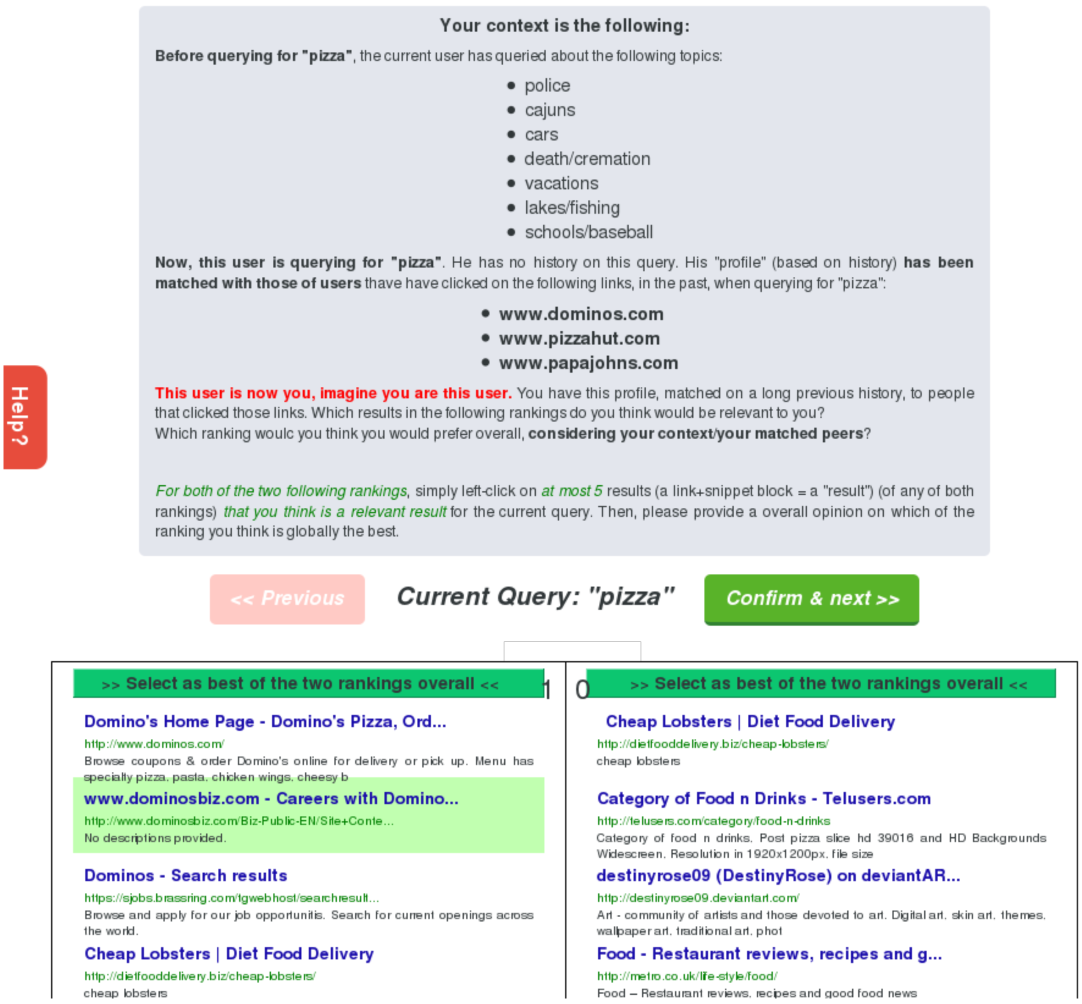
Figure 22: Example of results of the Evaluation in [6]

For every (query, user) pair, the volunteers are presented with the web UI shown in Figure 23, which contains:

- A summary of previous topics the “user” queried before issuing this query.
- URLs (3 maximum) on which “similar users” have clicked in the past.
- Two rankings, one on the left and the other on the right, randomly swapped at every page reload. One is the baseline, the other the CPPR ranking.
- Every ranking is composed of 10 “results”. A “result” is presented the same way it is on a modern search engine:
  - The title of the page (title extraction details in subsubsection 13.2.2.2).
  - The URL of the page (clickable).
  - A snippet with the content of the `meta` “description” tag (see subsubsection 13.2.2.2) or “no description provided” if the snippet is empty.

Volunteers are then asked to give the following judgments:

- Select “results” that they think are “relevant”. With a maximum of 5 relevant results *per ranking* and per query.
- Give an overall preference to one of the two rankings. Which one they think is the “best ranking overall”.



Your context is the following:

Before querying for "pizza", the current user has queried about the following topics:

- police
- cajuns
- cars
- death/cremation
- vacations
- lakes/fishing
- schools/baseball

Now, this user is querying for "pizza". He has no history on this query. His "profile" (based on history) **has been matched with those of users** thave have clicked on the following links, in the past, when querying for "pizza":

- **www.dominos.com**
- **www.pizzahut.com**
- **www.papajohns.com**

This user is now you, imagine you are this user. You have this profile, matched on a long previous history, to people that clicked those links. Which results in the following rankings do you think would be relevant to you?
Which ranking woulc you think you would prefer overall, **considering your context/your matched peers**?

For both of the two following rankings, simply left-click on *at most 5* results (a link+snippet block = a "result") (of any of both rankings) *that you think is a relevant result* for the current query. Then, please provide a overall opinion on which of the ranking you think is globally the best.

<< Previous
Current Query: "pizza"
Confirm & next >>
>> Select as best of the two rankings overall <<
1
0
>> Select as best of the two rankings overall <<

Domino's Home Page - Domino's Pizza, Ord...
http://www.dominos.com/
Browse coupons & order Domino's online for delivery or pick up. Menu has specialty pizza, pasta, chicken wings, cheesy b

www.dominosbiz.com - Careers with Domino...
http://www.dominosbiz.com/Biz-Public-EN/Site+Conte...
No descriptions provided.

Dominos - Search results
https://sjobs.brassring.com/tgwebhost/searchresult...
Browse and apply for our job opportunitis. Search for current openings across the world.

Cheap Lobsters | Diet Food Delivery
http://dietfooddelivery.biz/cheap-lobsters/
cheap lobsters

Cheap Lobsters | Diet Food Delivery
http://dietfooddelivery.biz/cheap-lobsters/
cheap lobsters

Category of Food n Drinks - Telusers.com
http://telusers.com/category/food-n-drinks
Category of food n drinks. Post pizza slice hd 39016 and HD Backgrounds Widescreen. Resolution in 1920x1200px. file size

destinyrose09 (DestinyRose) on deviantAR...
http://destinyrose09.deviantart.com/
Art - community of artists and those devoted to art. Digital art. skin art. themes. wallpaper art. traditional art. phot

Food - Restaurant reviews, recipes and g...
http://metro.co.uk/life-style/food/
Food – Restaurant reviews. recipes and good food news

Figure 23: Example Page of the User Study Web UI

Volunteers do the user study on their own without any in-person advice, as they are all doing it remotely. A help page is however displayed at the beginning of the user study, explaining what is the goal of the user study, what are the tasks they have to fulfill and a visual tutorial of the UI.

Volunteers answer on a (query, context) pair basis, and change page between each one, so that they are not displayed with all the rankings at once and not confused. They are, however, allowed to go back and forth in the user study in order to edit their previous answers in case they realize they misunderstood an instruction or any other reason. Note that, as they were not actively monitored while doing so, we cannot state whether volunteers tried to make all of their judgments unified or not, but they were not asked anything like this so we believe they did not try to.

One important difference with [6] is that users are able to browse the web pages that are presented in the results. They can click on links in order to make sure what they judge relevant or not. This is also explained in the user study. We also decide to use only 5 queries so that volunteers take all the time they need to make thourough choices.

18.1.2 Measures

In terms of measures, our user study features the two same measures as [6], in principle: precision and ranking preference.

However, the ranking preference is going to be presented in percentage of users who selected a ranking versus the other one, not in terms of "ranking preferred by majority" as the latter is not a precise measure in our opinion, although we will also display an additional table similar to Figure 22 in order to ease cross-work comparisons.

The precision is calculated differently in our case. Indeed, a problem arose with the way [6] calculated their precision (described in subsection 18.1.1): What happens if both rankings return the same sets of items, simply ranked differently or if volunteers consider that all items are relevant? Then, with the method used by [6] there is no possible comparison anymore. The precision would just be 1.0 for both ranking and no conclusion could be drawn. Looking at the preliminary results we had before launching the user study, we thought this case might well happen. Indeed, our web crawl was quite focused and the IR engine works relatively well (while it might have not at the time [6] was written which might explain their choice), and in addition to this, both rankings are often composed of the same set of items, ranked differently. So the possibility that volunteers select all results is not nil, and it is even quite probable that they select the same results on both rankings. This would both ruin the experiment's output information and would also be extremely tedious for the volunteers: it would mean 20 clicks for each query. This is why volunteers are asked to only select at most 5 relevant results per ranking, for every query. This way, we are going to define the *precision* $p_a(q, u)$

for the ranking a (baseline, cppr), a query q and a volunteer u is as follows:

$$p_a(q,u) = \frac{\sum_{r \in \mathcal{R}(q,u)} (11 - rank(r))}{\sum_{i \in [[1,5]]} (11 - i)}$$

with $\mathcal{R}(q,u)$ the set of results selected as relevant for the query q by the volunteer u and $rank(r)$ its rank in the ranking (1 = first (top) position, 10=last (bottom) position). Once the precisions for a given query are computed for all volunteers, we simply take the average over the volunteers.

However, this suffers from one inherent issue: Some volunteers could have generally selected a higher number of relevant results than others. Indeed, some volunteers could simply generally have a higher standard of "relevancy" and some others a lower one. The global averaged precision is calculated as if everyone had the same judgments standards, which is not necessarily true. As a consequence, we think the best comparison is the average *volunteer-specific precision difference* between the baseline and the CPPR rankings, for every query. The precision *difference* is calculated by subtracting the precision given to the *baseline* ranking by a given volunteer on a given query, to the precision computed for this same query and volunteer. Thus, if a volunteer generally has a higher standard of relevancy, she should have also selected a lower number of relevant results for the *baseline* ranking and thus the difference should be comparable volunteer-wide. Note that we use the *relative* difference here. The formula is as follows:

$$pd(q,u) = \frac{p_{cppr}(q,u) - p_{baseline}(q,u)}{p_{baseline}(q,u)}$$

Note that $pd(q,u)$ can be negative, which would mean that the baseline was more precise than the CPPR ranking.

A second set of measures is also done as a part of the evaluation. As in the original Topic-Sensitive PageRank paper [6] and in other research papers about PPR, we will compute the O-Similarity and the K-Similarity of the rankings. Such measures have been widely used in the past to compare rankings, and especially personalized PageRank-based ones. One can find them in [6] [75] [45] [7].

OSim indicates the degree of overlap between the top n URLs of two rankings. Formula:

$$OSim(r_1, r_2) = \frac{A \cap B}{k}$$

with k the size of both sets A and B which respectively represent the set of URLs of rankings r_1 and r_2

KSim, on the other hand, is the probability that two rankings agree on the relative ordering of a randomly selected pair of distinct nodes (u,v). Formula:

$$KSim(r_1, r_2) = \frac{|\{(u,v) : \ r_1', r_2' \ have \ same \ ordering \ of \ (u,v) \ and \ u \neq v\}|}{|U|^2 - |U|}$$

with $U = A \cup B$ and r_1' is the ranking constructed by adding items in r_2 but not in r_1 at the end of r_1 with their ordinal ranking from r_2 (that is to say, a rank equal to the sum of the highest rank in r_1 and their original rank in r_2). r_2' is constructed the same way. This will provide information on the amount of changes that our personalization brings to the final ranking.

Results of the user study using such measures are presented in section 18.2.

18.1.3 Selected Queries

The following methodology was used to select the queries used in the user study:

1. We ran a click analysis on the queries of the logs. For each query, we calculated:

   - Its total number of clicks.
   - Its click entropy, using the click entropy formula of [20].

2. We sorted the items descending order of click entropy and kept the 10,000 top ones.

3. We normalized number of clicks and click entropy values and computed the sum of their normalized values.

4. We sorted items in descending order of this sum.
5. We removed the queries with null clustering vectors (section 7.2).
6. We ran a pattern-based filter to get rid of a set of keywords and expressions not suitable for the user study (sex and pornography related queries).
7. We kept the first 1,000 of the resulting list.
8. We then manually randomly selected queries that would be easily understood by the volunteers of the user study and where personalization made sense (ambiguous or "general enough" queries).

The resulting list of queries is the following (displayed in this order in the user study):

1. "pizza"
2. "massage"
3. "water fountain"
4. "tattoo art"
5. "science experiments"

The corresponding users from the logs were selected by executing the `find_one()` method of MongoDB on the collection of clicks that had been stored at an earlier stage of the work. All selected users had queried the corresponding query and clicked at least once on one of its links, no other criteria was used.

The reason why we selected queries that way, especially the descending order sorting using the combination of clicks and click entropy is based on a mix of data requirements and previous works results:

- The most clicks we had, the highest was the probability that the "remapping" step (chapter 14) would not return an empty set of elements (that would mean we cannot personalize this query using our data).
- The higher the click entropy the most impact personalization can make, as a high click entropy means that different users do this query for different information needs, it has been experimented in [20].

18.2 Results

18.2.1 Ranking Preferences

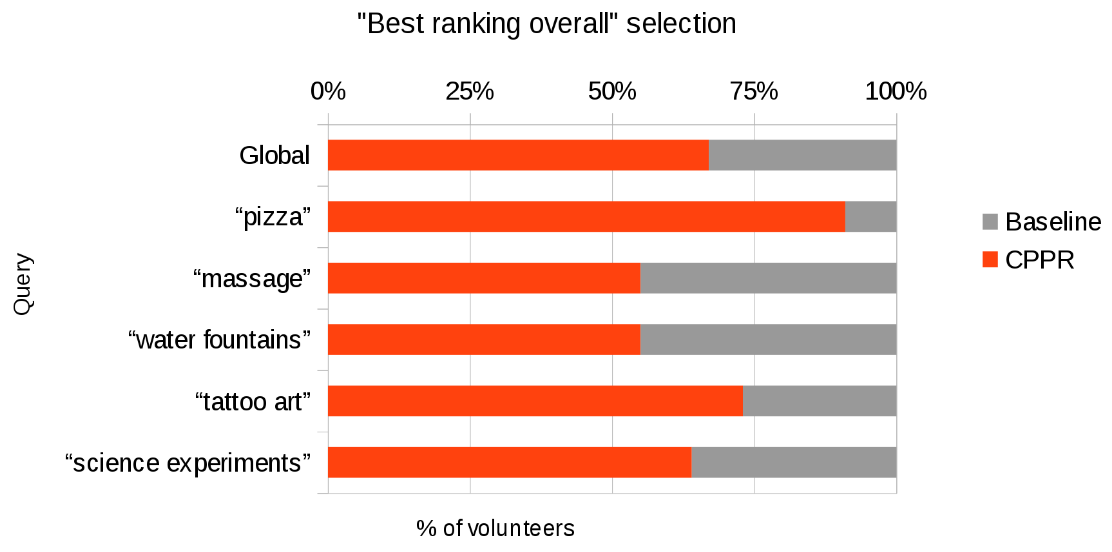


Figure 24: User Study: Volunteers Ranking Preferences

Figure 24 presents the volunteers ranking preferences for each query. As we can see, for all five queries, the majority of volunteers preferred the CPPR ranking. For two specific queries, preference towards the personalized ranking is over 70%. Also, only one volunteer did not prefer the CPPR ranking for the first query.

As a way to ease comparison with [6], we summarize the preferences in a similar table to Figure 22 in Table 1.

Table 1: Preferred Ranking Table, in comparison with Figure 22

Query	Preferred By Majority (>50%)
pizza	CPPR
massage	CPPR
water fountain	CPPR
tattoo art	CPPR
science experiments	CPPR
Globally	CPPR

From preferences presented in Figure 24 and Figure 22, we can already say that the experiment shows that users perceive a *measurable* improvement thanks to the personalization. This is an encouraging result. Indeed, the main goal of a widely-deployed end-user system such as a search engine is not only to theoretically improve its results from a scientific point of view but also from an applied point of view to satisfy its users.

18.2.2 Precision

The average precisions of both baseline and CPPR rankings can be found in Figure 25 and are in accordance with the percentages of preferences described in Figure 24 except for the third query where, surprisingly and while a majority of users still preferred the personalized ranking, they consistently selected relevant results that were ranked higher in the baseline in comparison with the CPPR ranking. This might mean that the personalized ranking for this query, overall, contained more relevant elements, even if the 5 most relevant ones were ranked lower, and volunteers globally preferred a ranking that has more relevant results even if they are not ranked optimally. However, as they could not select more than 5 relevant results, they could not express the fact that more relevant results were present in the personalized ranking.

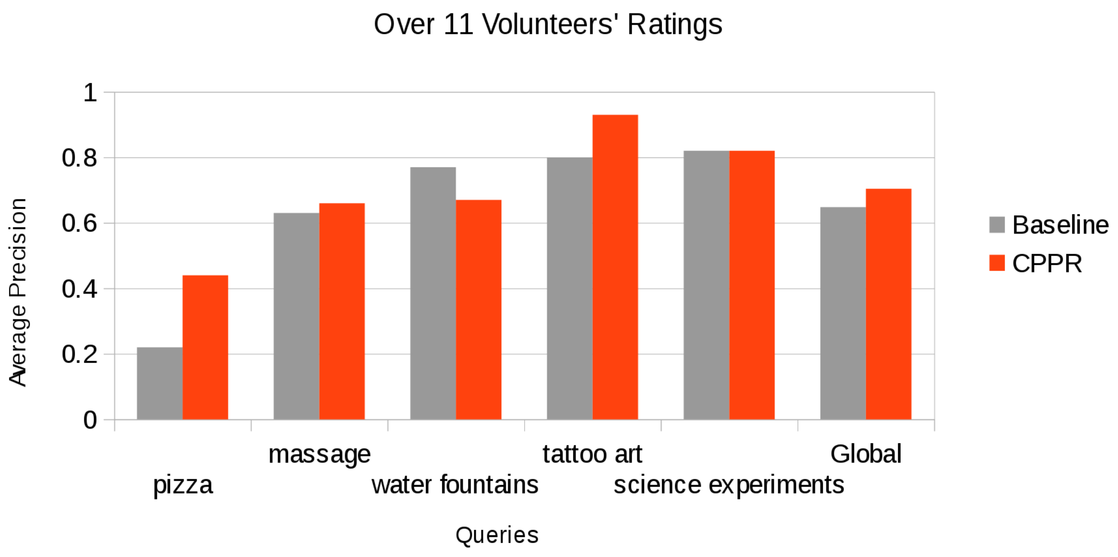


Figure 25: Average Precisions Of Baseline And CPPR Rankings, Per Query

The *volunteer-wise* relative precision difference, that is to say the relative precision difference computed on a per- volunteer basis, is shown in Figure 26. An issue arose when computing the precision difference using the formula proposed in subsection 18.1.2: For the query “pizza”, nearly half of the volunteers did not select any relevant results for the baseline, effectively assigning it a nil score. But those same volunteers did select at least one relevant result for the personalized ranking. As a consequence, the *relative* difference for such volunteers is $+\infty$, which obviously drives the *average* over all volunteers to be $+\infty$ too. In order to overcome this problem, we observed that in all such cases volunteers in fact simply selected no results for the baseline and one for the personalized ranking. We considered that assigning a relative difference of 100% would be a fair approximation. Indeed, the volunteer selection was basically a binary judgment: either this result was there, or not. For the rest of the queries, the precision difference is pretty much following the trend of difference in average precisions that could be observed in Figure 25, with a higher amplitude. This means that volunteers globally did the user study with similar “standards of relevancy”. The average of the volunteer-wise relative precision difference is 21.60%. This is a very encouraging result as this means that globally, volunteers indeed felt a significant improvement in the personalized ranking, compared to the baseline.

18.2.3 OSim and KSim

Figures Figure 27 and Figure 28 present the resulting OSim and KSim measures as defined in subsection 18.1.2.

As we can see in Figure 27, the second query has a nil OSimilarity for the top 5 results, while all the other queries have non-nil similarities. This is probably related to the fact that the precision of the second query is the only one that was worse in the CPPR ranking compared to the baseline one.

The measures overall are similar to previous works using such measures, such as [6] and [7], which means that our personalization algorithm performs on similar grounds compared to previously proposed ones, even though our experimental setup is suboptimal. This is, once more, a very encouraging result for future work in this area.

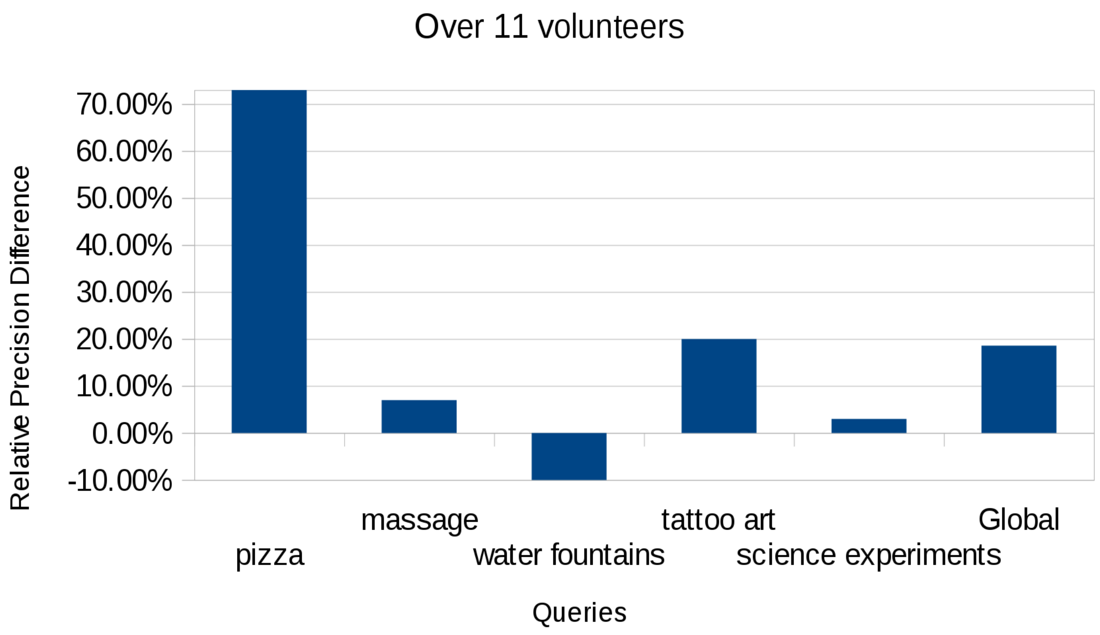


Figure 26: Volunteer-Wise Relative Precision Difference From Baseline

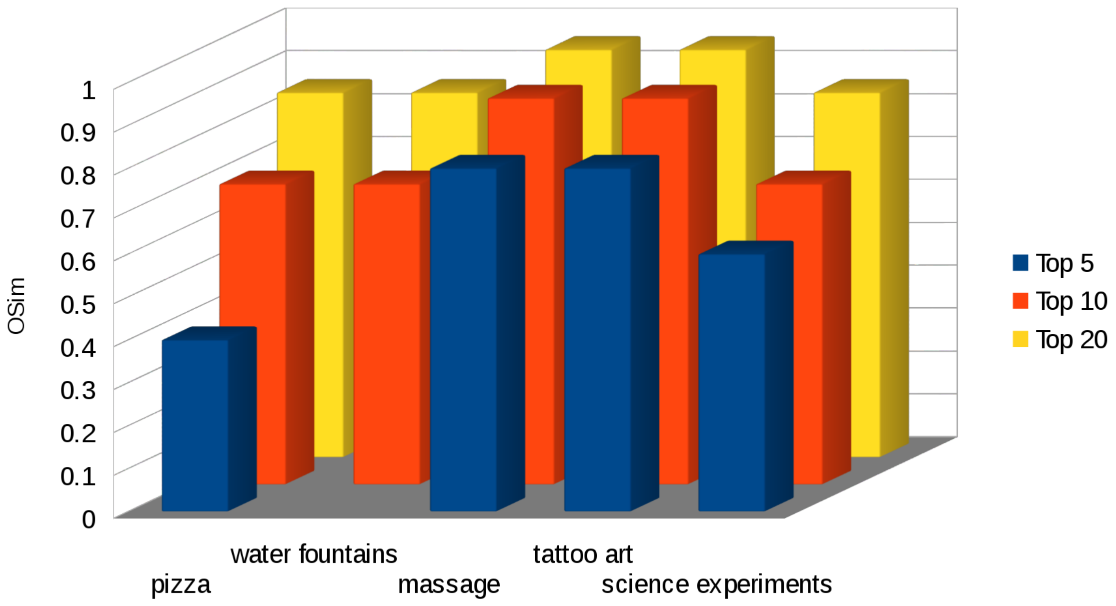


Figure 27: Per Query OSim for top 5, 10 and 20 results

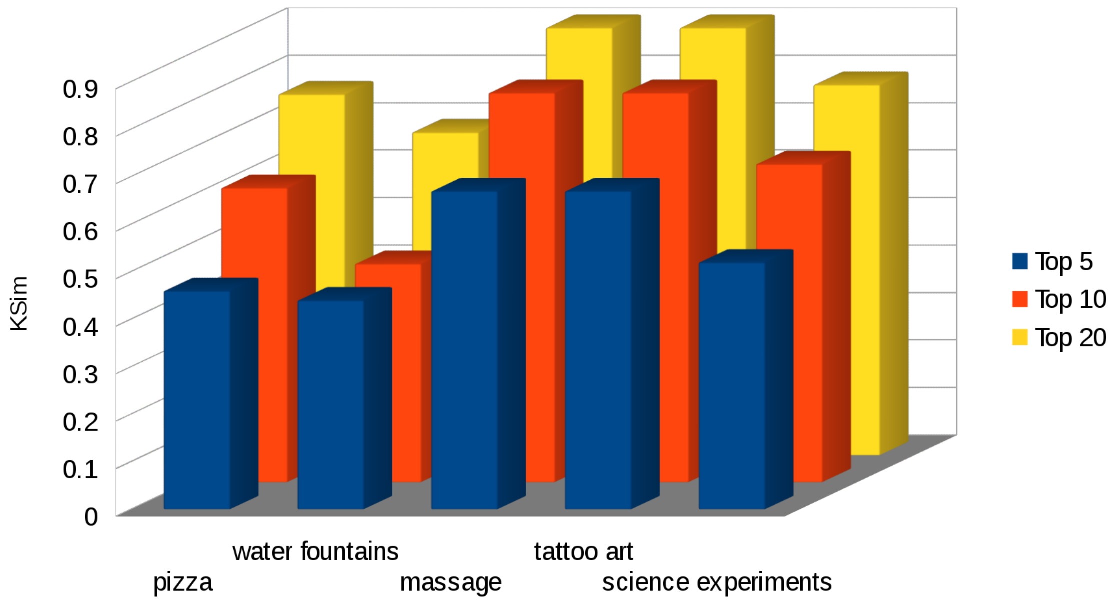


Figure 28: Per Query KSim for top 5, 10 and 20 results

Part V

Conclusion

19 Conclusion

In this Master Thesis, we have developed a personalization technique to be applied to web search using the Personalized PageRank as means of personalization. We have proposed a way to evaluate such technique using an online platform and a user study protocol, reusing evaluation features of previous works when it was possible and improving them where we thought they had issues or lacked precision. We have provided all necessary details of our experimental setup such that future works can both verify our results and compare against our implementation, that is available to all on Internet [49]. Results of our evaluation have showed that our approach works and has a measurable impact on the quality of the results.

In terms of the research questions that were formulated in section 1.2, we believed we answered the research questions in the following ways:

1. We can use the relative number of clicks on queries from the clicks logs. This is enough to be able to build a profile and to be integrated with an external source of semantics, provided by [21]'s results in our setup.
2. The user profile can be constructed using the clickthrough data analysis formulas presented in chapter 7. Based on the work that was done in [20], and together with our integration of semantics (section 7.2), it provides a good way to match users with each other to allow collaborative scores to be computed for web graph nodes. This, in return, allows personalization of the SERP through PageRank using the *personalization vector*.
3. Results show that the approach can perform well, even sometimes on a comparable scale with previous approaches (as [6] and [20]) even when the setup is not optimal.
4. We have proposed a rigorous user study protocol, as described in subsection 18.1.1, to evaluate web search personalization in the case of search results re-ranking through PageRank.

There still is room for improvement in many different parts of our approach though. As such, we will discuss in the next section the details of the improvements that we believe can be performed and the future work that can be done related to this technique.

20 Future Work

20.1 User profiling & Usage Mining

The user profiling method we use in our work is based mainly on previous simple user profile representations such as [20]. However, it has been proven that using multiple features, instead of basing the personalizationg on a unique feature (in our case, a semantic-aware clicks-based profile), can greatly improve the performance of the personalization system. In particular, [39] presents a unified framework for web search personalization and proposes an already quite complete list of personalization features that can be extracted from search engines clicks logs.

[39] and [33] also use a technique of "temporal decrease" of the importance of the personalization information. The older the information, the less importance it is given. This comes from the fact that the user is likely to have interests closer to what she was doing 5 minutes ago compared to what she did a month ago. However, long-term profiles are also useful especially in collaborative systems as their permit to match users together on a "similar minds" measure basis. Thus, it seems important to use both long-term and short-term information in order to provide with different personalization features that would help both personalized effectively and match users effectively. [39] shows that a combination of short and long-term information outperforms in all cases using only one, the other or using them without differentiating them. In the case of our technique, one could for instance use the long-term profile of the user in order to match her with similarly minded people. But once the matching would be done, we could either restrict the information used to constitute the web graph node personalization score to short-term information or use both short-term and long-term information with different weights.

An interesting approach to be studied in future work would be to use an adaptive personalization algorithm. Such an adaptive technique could look for short-term information, evaluate the level of personalization it would provide and, depending on the result, choose to use the long-term information too or restrict the personalization to scores based on short-term information. The same technique can be imagined for the matching too. We could even go one step further and simply constitute two profiles for every user: a long-term and short-term one. When the personalization system is run, depending on the quality of using one profile against the other, it dynamically chooses a profile. This would be, in some ways, a sort of combination of ideas presented in [39] (short/long-term profiles) and [16] (use of multiple profiles).

Another important feature that is not taken into account in our usage mining is the *dwell time* related to each click of the search engine logs. The *dwell time* is the estimated time that the user has spent on the page she has clicked. This time can be estimated using the other actions of the user with the search engine. [4] and [51] have shown that the dwell time is an important measure and allows to improve the recommendation/personalization performance.

20.2 CPPR Computation Scaling

A criticism that can be done to our experimental setup and more widely to our technique is that it is very expensive in terms of computation, thus making it not suitable to be deployed on a large scale.

The approach is indeed very expensive in *our* experimental setup. However, a great amount of work has been done on optimizing personalized PageRank computation in the past. Among the promising ones, [40], but also [41] and [42] allow to speed up significantly the computation of the Personalized PageRank. In particular, [40] presents a technique where a part of the PageRank vectors is only computed once and only the rest of the computation has to be run again in order to update the PageRank.

Moreover, adaptation of the PageRank algorithm to be run on the Google MapReduce system are available on the Internet[3]. A large scale system would mean that we also have access to larger scale computation networks. By com-

[3]https://github.com/castagna/mr-pagerank (last visited 14/09/2014)

bining the previously stated optimizations of PPR computation in addition to distributed computation techniques, we believe one can achieve CPPR computation on larger scales with an interesting ratio of effort/output. It is also to be considered that we ran the PageRank computation with a very high precision. It may not be necessary at all to have such a precision in the case of the CPPR ranking. Future work could especially study where precision parameter gives the best ratio in terms of needed resources compared to the outputted personalized ranks. It is not always necessary to have big difference in scoring in order to differentiate items, thus, finding the exact threshold at which we have enough differences to be able to make good recommendations, while still having a very low performance demand is an important step in the deployment of such a system in production and could be a future work on this technique.

20.3 Collaborative CPPR (CCPPR)

In section 9.4 we talk about the possibility of a two-level collaboration for the technique that we use. Two-level collaboration could probably yield interesting results but has the drawback of being heavier in terms of resources. Indeed, instead of computing one CPPR vector per user and query, one needs to compute n CPPR vectors with n the number of similar users to the current user. It would nevertheless be interesting to evaluate such a technique and especially to compare its results to the standard CPPR one.

In relation with the idea of having multiple profiles that we have suggested in the previous section, CCPPR could be a way to achieve this. One could imagine having several restricted-size sets of similar users, with a set being emphasized on a semantic category more than others and using a combination of CPPR using this limited number of users. One could also use this technique in order to bring *diversity* to the results by injecting users that are less similar but have for instance been proven to be experts in the domain and thus whose clicks provide higher relevancy to the current query.

Finally, one last application could be to select the set of CPPR vectors that we combine into a CCPPR vector not from the user profile but from a query profile, thus making the CCPPR vector query specific and not user specific anymore, in order, for instance, to globally improve the search results for users that do not have any profile yet.

Bibliography

[1] The Toxic Terabyte: How Data-Dumping Threatens Business Efficiency, IBM Global Technology Services, July 2006

[2] The PageRank citation ranking: Bringing order to the web. L Page, S Brin, R Motwani, T Winograd - 1999

[3] J. M. Carroll and M. B. Rosson. Paradox of the active user. Pages 80–111, 1987.

[4] F. Zhong, D. Wang, G. Wang, and W. Chen, Incorporating post-click behaviors into a click model

[5] Z. Dou, R. -Song, and J.-R. Wen, A large-scale personalized search strategies, 2007

[6] T. H. Haveliwala, Topic-Sensitive PageRank: A Context-Sensitive Ranking Algorithm for Web Search, vol. 15, no. 4, pp. 784–796, 2003.

[7] M. Eirinaki, M. Vazirgiannis, and D. Kapogiannis, "Web path recommendations based on page ranking and markov models," 2005, pp. 2–9.

[8] Schilit, B., Theimer, M. Disseminating Active Map Information to Mobile Hosts. IEEE Network, 8(5) (1994)

[9] Abowd, Gregory D., et al. "Towards a better understanding of context and context-awareness." Handheld and ubiquitous computing. Springer Berlin Heidelberg, 1999.

[10] Coutaz, Joëlle, et al. "Context is key." Communications of the ACM 48.3 (2005): 49–53.

[11] Matthias Baldauf. A survey on context-aware systems, Int. J. Ad Hoc and Ubiquitous Computing, Vol. 2, No. 4, 2007

[12] D. Preuveneers and J. Van Den Bergh, "Towards an extensible context ontology for ambient intelligence", Intell., 2004.

[13] Tao Gu; Pung, H.K.; Da Qing Zhang, “A middleware for building context-aware mobile services”, 2004

[14] P. Fahy and S. Clarke. CASS - middleware for mobile context-aware applications. In Proc. Mobisys 2004 Workshop on Context Awareness, 2004.

[15] Robert Armstrong, Dayne Freitag, Thorsten Joachims, and Tom Mitchell. WebWatcher: a learning apprentice for the world wide web. In 1995 AAAI Spring Symposium on Information Gathering from Heterogeneous Distributed Environments, pages 6–12, 1995.

[16] Kurt D. Bollacker, Steve Lawrence, and C. Lee Giles. A system for automatic personalized tracking of scientific literature on the web. In Digital Libraries 99 - The Fourth ACM Conference on Digital Libraries, pages 105–113. ACM Press, 1999.

[17] Hyoung R. Kim and Philip K. Chan. Learning implicit user interest hierarchy for context in personalization. In Proceedings of the 8th international conference on Intelligent user interfaces, IUI ’03, pages 101–108, New York, NY, USA, 2003. ACM.

[18] Xuehua Shen, Bin Tan, and Chengxiang Zhai. Implicit user modeling for personalized search. In Proceedings of CIKM, 2005.

[19] Fabian Abel, Qi Gao, Geert-Jan Houben, and Ke Tao. Semantic enrichment of twitter posts for user profile construction on the social web. In Proceedings of the 8th extended semantic web conference on The semantic web: research and applications - Volume Part II, ESWC’11, pages 375–389, Berlin, Heidelberg, 2011. Springer-Verlag.

[20] Z. Dou, “A Large-scale Evaluation and Analysis of Personalized Search Strategies,” pp. 581–590, 2007.

[21] L. Limam, “Usage-Driven Unified Model for User Profile and Data Source Profile Extraction”, 2014.

[22] F. Liu and C. Yu, “Personalized Web Search by Mapping User Queries to Categories”, 2002.

[23] M. Pazzani, “Learning and Revising User Profiles : The Identification of Interesting Web Sites”, 1997.

[24] James Pitkow et. al., Personalized search. Commun. ACM 45, 9, 2002

[25] Alfred Kobsa and Wolfgang Pohl. The user modeling shell system BGP-MS. User Modeling and User-Adapted Interaction, 4(2):59–106, 1994.

[26] Stuart E Middleton, N. R Shadbolt, and D. C De Roure. Ontological user profiling in recommender systems. ACM Transactions on Information Systems (TOIS), 22(1):54–88, January 2004.

[27] M. Eirinaki, M. Vazirgiannis.: Web Mining for Web Personalization ACM Transactions on Internet Technologies (ACM TOIT). Vol.3

[28] S. W. Ave, “Effective Personalization Based on Association Rule Discovery from Web Usage Data.”

[29] J. Li and O. R. Za, “Using Distinctive Information Channels for a Mission-based Web Recommender System.”

[30] M. Balabanovic and Y. Shoham, “Fab: Content-Based, Collaborative Recommendation,” Comm. ACM, vol. 40, no. 3, pp. 66–72, 1997.

[31] N. Littlestone and M. Warmuth, “The Weighted Majority Algorithm,” Information and Computation, vol. 108, no. 2, pp. 212–261, 1994.

[32] D. Goldberg, D. Nichols, B.M. Oki, and D. Terry, “Using Collaborative Filtering to Weave an Information Tapestry,” Comm. ACM, vol. 35, no. 12, pp. 61–70, 1992.

[33] J. Schlötterer, C. Seifert, and M. Granitzer, “Web-based Just-In-Time Retrieval for Cultural Content.”

[34] Breese, John S., David Heckerman, and Carl Kadie. “Empirical analysis of predictive algorithms for collaborative filtering.” Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence. Morgan Kaufmann Publishers Inc., 1998.

[35] Y.H. Chien and E.I. George, “A Bayesian Model for Collaborative Filtering,” Proc. Seventh Int’l Workshop Artificial Intelligence and Statistics, 1999.

[36] L. Getoor and M. Sahami, “Using Probabilistic Relational Models for Collaborative Filtering,” Proc. Workshop Web Usage Analysis and User Profiling (WEBKDD ’99), Aug. 1999.

[37] D. Pavlov and D. Pennock, “A Maximum Entropy Approach to Collaborative Filtering in Dynamic, Sparse, HighDimensional Domains," Proc. 16th Ann. Conf. Neural Information Processing Systems (NIPS ’02), 2002.

[38] Mark J. Carman, Fabio Crestani, Morgan Harvey, and Mark Baillie. Towards query log based personalization using topic models. In Proceedings of the 19th ACM international conference on Information and knowledge management, CIKM ’10, pages 1849–1852, New York, NY, USA, 2010. ACM.

[39] P. N. Bennett, R. W. White, W. Chu, S. T. Dumais, P. Bailey, F. Borisyuk, and X. Cui, “Modeling the Impact of Short- and Long-Term Behavior on Search Personalization,” 2012.

[40] Jeh, G., Widom, J.: Scaling personalized Web search. In: Proc. 12th International World Wide Web Conference. (2003)

[41] Haveliwala, T.: Efficient computation of pagerank. Technical report, Stanford Database Group (1999)

[42] Kamvar, S.D., Haveliwala, T.H., Manning, C.D., Golub, G.H.: Exploiting the block structure of the Web for computing PageRank. Technical report, Stanford University (2003)

[43] A. Hotho, J. Robert, C. Schmitz, and G. Stumme, “FolkRank : A Ranking Algorithm for Folksonomies”, pp. 2–5.

[44] “DMOZ, The Open Directory”, http://dmoz.org/, last visited 14 Sept. 2014

[45] Optimizing search and ranking in folksonomy systems by exploiting context information. By F. Abel, N. Henze, and D. Krause. Lecture Notes in Business Information Processing, volume 45(2), Springer, 2010.

[46] Bao, S., Xue, G., Wu, X., Yu, Y., Fei, B., Su, Z.: Optimizing Web Search using Social An- notations. In: Proc. of 16th Int. World Wide Web Conference (WWW 2007), pp. 501–510. ACM Press, New York (2007)

[47] F. Abel, M. Frank, N. Henze, D. Krause, D. Plappert, and P. Siehndel, "GroupMe ! - Where Semantic Web meets Web 2.0", no. 3.

[48] "KDD Cup 2005: Internet user search query categorization", http://www.sigkdd.org/kdd-cup–2005-internet-user-search-query-categorization, last visited 14 Sept. 2014

[49] "Collaborative(ly) Personalized PageRank", Théo Dubourg, https://github.com/tdubourg/collaborative-personalized-pagerank (publicly available starting Dec. 2014)

[50] Greg Pass, Abdur Chowdhury, and Cayley Torgeson. A picture of search. In Proceedings of the 1st international conference on Scalable information systems, InfoScale 2006, New York, NY, USA, 2006. ACM.

[51] "Data - Personalized Web Search Challenge | Kaggle", http://www.kaggle.com/c/yandex-personalized-web-search-challenge/data, last visited 14 Sept. 2014

[52] "Sogou laboratory data download - user query logs", http://www.sogou.com/labs/dl/q.html, last visited 14 Sept. 2014

[53] Kalervo Jarvelin, Jaana Kekalainen: Cumulated gain-based evaluation of IR techniques. ACM Transactions on Information Systems 20(4), 422–446 (2002)

[54] Kaggle Yandex "Personalized Search Challenge", 2014, https://www.kaggle.com/c/yandex-personalized-web-search-challenge

[55] "Laboratory for Web Algorithmics", http://law.di.unimi.it/datasets.php, last visited 14 Sept. 2014

[56] Hirai, Jun, et al. "WebBase: A repository of web pages." Computer Networks 33.1 (2000): 277–293., http://diglib.stanford.edu:8091/∼testbed/doc2/WebBase/, last visited 14 Sept. 2014

[57] "CommonCrawl", Common Crawl Organization, http://commoncrawl.org/, last visited 14 Sept. 2014

[58] "Common Crawl Corpus", http://aws.amazon.com/datasets/41740, last visited 14 Sept. 2014

[59] "80 terabytes of archived web crawl data available for research", http://blog.archive.org/2012/10/26/80-terabytes-of-archived-web-crawl-data-available-for-research/, last visited 14 Sept. 2014

[60] "The ClueWeb09 Dataset", Lemur Project, http://lemurproject.org/clueweb09.php/index.php#Services, last visited 14 Sept. 2014

[61] "Scrapy | An open source web scraping framework for Python", Scrapy Project, http://scrapy.org, last visited 14 Sept. 2014

[62] "Terrier IR Platform V4.0", University Of Glasgow, http://terrier.org/, last visited 14 Sept. 2014

[63] "Ultra-fast Search Library and Server", http://lucene.apache.org/, last visited 14 Sept. 2014

[64] "Apache Solr", http://lucene.apache.org/solr/, last visited 14 Sept. 2014

[65] "Elasticsearch.org Open Source Distributed Real Time Search", ElasticSearch, http://elasticsearch.org/, last visited 14 Sept. 2014

[66] "Apache Nutch", http://nutch.apache.org/, last visited 14 Sept. 2014

[67] "Beautiful Soup: a library designed for screen-scraping HTML and XML.", http://www.crummy.com/software/BeautifulSoup/, last visited 14 Sept. 2014

[68] "Chardet: The Universal Character Encoding Detector", https://github.com/chardet/chardet, last visited 14 Sept. 2014

[69] "The OpenMP API specification for parallel programming", http://openmp.org/wp/, last visited 14 Sept. 2014

[70] "PEP 1 – PEP Purpose and Guidelines", http://legacy.python.org/dev/peps/pep–0001/, last visited 14 Sept. 2014

[71] “PEP 20 – The Zen of Python”, http://legacy.python.org/dev/peps/pep–0020/, last visited 14 Sept. 2014

[72] J.C.Borda. Mémoire sur les élections au scrutin. Histoire de l’Académie Royale des Sciences, 1781.

[73] “Borda Count”, http://en.wikipedia.org/wiki/Borda_Count, last visited 14 Sept. 2014

[74] Peter Tannenbaum, “The Borda Count Method”, “Excursions in Modern Mathematics”, Chap 1.3, 2010

[75] Abel, Fabian, et al. “Semantic enhancement of social tagging systems.” Web 2.0 & Semantic Web. Springer US, 2009. 25–54.