Collaboratively Searching the Web – An Initial Study

Agustin Schapira
Center for Intelligent Information Retrieval
Computer Science Department
University of Massachusetts, Amherst
Amherst, MA 01003 USA
schapira@cs.umass.edu

Abstract: We investigated the hypothesis that precision in web search engines can be improved by using relevance information provided by a large number of different users. We tested the hypothesis with a search engine that monitored which documents were selected among the results for a query and then used that information to re-rank the results when other users submitted the same query. The engine ran on the WWW and received nearly 3,000 queries in a period of two months. An analysis of the performance history of the engine does not show any improvement in the quality of the results returned for very frequent queries. Anecdotal evidence, however, suggests that for less popular queries the engine did learn to re-rank documents in such a way that relevant documents appear at the top of the results list. More experimentation and especially more data are needed to prove or disprove the hypothesis.

Introduction

In a recent study [1], Lawrence and Giles estimate the size of the World Wide Web at 320,000,000 pages. Web search engines help users find information in this vast repository. Although they provide an excellent start, Internet search engines have limited success as information finders or filters. Users typically complain about a) the enormous quantity of results returned for their query, and b) the fact that usually a large percentage of the documents returned have no relevance to their query. For instance, a search for "*women's world cup soccer*" submitted to Infoseek while the tournament was being played returned a) over 45 million documents, and b) a list in which the top 10 results include –in this order– 2 relevant pointers to sites with up-to-date scores and statistics, 3 news documents that were published a year before the tournament started, 1 pointer to a related tournament, 1 pointer to a current newspaper article, 3 completely irrelevant results (including a page about the men's world cup in France last year and a minor soccer league in Texas), and no pointer to the official site for the tournament at http://wwc99.fifa.com/. Only four documents in the top-10 list were relevant.

This paper investigates the efficacy of one particular method of improving result quality in Internet search engines. The method consists in using *implicit relevance feedback* gathered from the users to enhance the standard Information Retrieval techniques applied in search engines. Concretely, the idea is to monitor which documents the users choose from the results list; this *implicit relevance feedback* information is then fed into the system and used to rank the results the next time that the same query is submitted –maybe by a different user. Documents that are more frequently chosen get an increment in their

score (initially computed with standard IR techniques), and documents that are very rarely picked see a reduction in their score. The objective of this scoring mechanism is to generate results of increasing quality over time: as more users select documents from the results, the *chosen* ones will get higher scores and thus will move to the top of the list while *unpopular* documents will see their score diminish and will thus migrate to the bottom of the results list. In the women's soccer example, we would expect this scoring mechanism to help solve the second problem, the low quality of results in the top 10 documents. We would expect users to consistently select those documents that contain up-to-date focused information about the World Cup and to consistently ignore the irrelevant documents such as, for example, the one about the Texan league. After receiving this kind of user feedback, the system should be able move the relevant documents to the top of the list and give them a very high score, and to move down or even eliminate the clearly irrelevant documents. This method, of course, will not help in retrieving **the one** relevant document (the official website for the event), because that document was never retrieved by the standard IR mechanism in the first place. Regarding the other problem, the very large amount of documents retrieved, this method could only help solve it indirectly by showing results of very high quality at the top of the list and thus eliminating the need to look any further than the 10 or 20 documents that most search engines show in the first results page.

The success of such a mechanism depends on the validity of several hypotheses and assumptions. In the first place, for this system to succeed there should be an overlap in the queries that a search engine receives; if every user types a different query, there is not much space for learning. Another hypothesis is that we can in general infer the relevance or irrelevance of a document by monitoring whether the user decides to visit it (and click on it) or not. Obvious questions arise: what if the user clicks on a document and then returns immediately because it is not relevant? Maybe the time a user spends reading a page is a better estimate. What if the user already knows a particular document and then does not select it? Does it mean that the document is not relevant? A third, and very important hypothesis, is that all users (or at least the majority) have the same thing in mind when they submit the same query. That might not be the case: the query "*Miami dolphins*" probably refers to two different things if it is submitted by a football fan or by an expert in the field of animal communication working in Florida.

To test these hypotheses, we first studied the patterns of queries submitted to a very popular search engine to verify the validity of the first hypothesis, and then built a new search engine that implements the described scoring mechanism. This paper describes the implementation of the system and discusses the anecdotal results obtained with a preliminary experiment.

Related work

Some of the ideas in this project are based on the progress in the area of *recommender systems*, or *collaborative filtering*. Resnik and Varian [2] define recommender systems as computer systems that receive recommendations as inputs and then aggregate those

inputs and direct the recommendations to appropriate recipients. An example of this type of system is MovieLens, a web-based recommender system that asks users how they liked a list of movies and then computes a measure of similarity in tastes among different users. With those movie ratings and the similarity measures, MovieLens can recommend to each user new movies that other users with similar interests have enjoyed. In an environment like this, users make public their rankings and opinions and thus collaborate indirectly with other users, hence the name "collaborative filtering".

Recommender systems need very large databases of rankings/tastes to be able to compute similarity metrics among users. A World Wide Web search engine is in a sense an ideal platform for a recommender system, since data abounds in those environments. A popular search engine like Excite receives queries in the order of 1,000,000 a day. With such a large number of visits there is a big potential for using information provided by one user to improve the quality of another user's interaction with the search engine.

Recommender systems are known to perform well with recommendation items for which they have received many rankings, and poorly otherwise. This implies that a large overlap in the queries is necessary in order to enhance a search engine's result with collaborative filtering techniques. In an unpublished study, we have investigated a log of one million queries submitted to Excite on a single day. Our results show a very large overlap in the queries. The queries follow a very skewed distribution, with a few queries appearing many times and thus representing a large percentage of all the searches, and a large number of queries that only appear once. In the particular day when the Excite log was recorded, only 641 queries (0.1% of all the unique queries) account for more than 10% of all the queries received. These results are similar to the ones reported by Jansen et. al.[3]. Taken together, those studies support the idea of using techniques borrowed from recommender systems in the realm of Internet search engines.

The ideas presented in this paper also borrow from existing techniques to improve the quality of IR systems. One of those techniques is relevance feedback [4], a method that consists in presenting a set of results to the user and asking him to judge their relevance in respect to his query. The system then uses that *relevance feedback* to add and remove terms to the original query; terms to add are taken from documents judged relevant, and terms to remove are taken from irrelevant documents. The improved query is re-run and the new results presented to the user, who can provide even more relevance feedback; the process loops until the user feels satisfied with the quality of the results. The relevance feedback mechanism has been shown to produce results of increasing quality [5]. It has been widely used in traditional (non web-based) IR systems, where users have a precise information need and are willing to spend time with the system to improve the query in order to get high quality results.

In Web-based environments, however, users are not necessarily willing to spend time refining their searches. In a study of the usage of Internet search engines, Jansen et al. [3] show that a very small percentage of people (less than 5%) used the relevance feedback mechanism provided by Excite, even though it generates better results. Their study does not investigate the causes of such low usage of the relevance mechanism. However, they

do report other behaviors among Excite users that, taken together, describe a generalized attitude among the users of search engines. For instance, the majority of the queries that Excite received had only 2 or 3 terms, very short compared to the length of queries in more traditional IR systems (some Internet search engines even perform poorly with long queries, effectively discouraging lengthy queries). In addition, a very small percentage of the users scrolled to see returned documents beyond the first page of results, and a very small number of times did individual users refine their original queries if they could not find what they were looking for in the first iteration. In other words, the study by Jansen et al. shows (without speculating about causes) that users tend to interact very little with Internet search engines, at least in comparison to what is typical in more traditional IR systems.

The tendency for users to have little interaction with Web search engines encourages the use of ***indirect measures*** to implement relevance feedback mechanisms in those engines. The challenge is to find indicators that, on the one hand, do not require the users' interaction and that, on the other hand, provide good clues as to which documents are relevant and which are not. The number of clicks that a document receives is, in some sense, an indirect indicator of its relevance. If a user chooses a particular document from a list of possible relevant pages, then it seems natural to increase one's expectation that the document is relevant to the user's query. A better indicator could be the time that a user spends reading a document, but it is more difficult to implement and implies a larger network overhead. It has also been suggested by Lieberman [6] that a page whose links are followed by the user is likely to contain relevant information. Finally, Lieberman claims that knowing that a user has bookmarked a page is a very good indicator that the page is *very* important.

It is important to note here that the fact that relevance information has to be gathered indirectly makes the task of building a collaborative search engine much harder than that of building a traditional recommender system. Systems such as MovieLens receive opinions directly from the users and have repeated interactions with each user, whereas in a collaborative Internet search engine the interaction is not very frequent and the users' opinions have to be gathered from indirect indicators. Furthermore, ambiguity in an Internet search engine is an issue not present in a recommender system: there is no ambiguity about what movie the title "Casablanca" refers to, while it is less clear what "Casablanca" means as a user query in a search engine.

This paper is also an attempt to analyze the effectiveness of ideas that have already been implemented in a commercial product, DirectHit [8]. DirectHit is an add-on to Internet search engines that measures document popularity in the context of individual queries that the engines receive. DirectHit monitored document selections from results lists for almost a year, and then started to use that information to compute the score of documents for previously-seen queries. DirectHit has not made available any information about their method or their evaluation of the engine, so one can only speculate about the specific algorithms they use and their effectiveness.

Methodology

Hypotheses

The success of a recommender search engine system that gathers relevance information indirectly by monitoring document clickrate and then uses that information to generate higher-quality results for all users who type the same query depends heavily on the validity of several hypothesis:

1. There is some overlap in the queries that the search engine will receive.
2. The relevance or irrelevance of a document can be inferred with acceptable accuracy by monitoring whether the user decides to visit it (and clicks on it) or not.
3. There is a considerable overlap in the subset of documents that all users (or at least the majority) pick from a list of results for a given query. That is, almost all the users who type the same query will find the same subset of documents relevant.

The first hypothesis has to be true for any recommender system to be successful. We have verified this hypothesis in the case of Internet search engines by analyzing a log of a million queries submitted to Excite on a single day. Data on that log led us to the conclusion that the queries follow a very skewed distribution: a small group of queries are repeated a very large number of times and thus account for a large percentage of the total number of queries, while a very large number of queries are only seen once or twice in a day. We also continued the Jansen et al. study [3] and investigated the distribution of the queries according to topics. We trained a neural network to group queries according to the similarities of the documents that they retrieved, and we found that a very large percentage of the Excite queries fall into two or three popular subjects (pornography, computer-related information, intellectual subjects such as universities and museums, etc.) while a number of categories receive very few queries. These results suggest that the use of a recommender system could noticeably improve the quality of the results for queries that are submitted by several people, while no improvement should appear for the queries that are submitted only once, due to the lack of datapoints to learn from.

The second hypothesis deals with the acquisition of relevance judgements by indirect means (indirect in the sense that the users do not provide that information explicitly). In this study, we have not tried to validate the hypothesis that document selection is a good indicator of relevance, but instead have taken it as an assumption for the rest of the system. There are several situations in which this assumption proves wrong; examples include cases where users select a document by mistake (they meant to select another one), cases where users select a document and realize that it is non-relevant as soon as the page is loaded, cases where users do not select a document because its title and description leads them to believe that it is not relevant, cases where users do not select a relevant document because they have already seen it, etc. In all these cases, monitoring document selection does not lead, by itself, to an accurate measure of document relevance. More advanced measures might provide better estimates, such as monitoring the time a user spends reading a document or detecting how many links a user follows

from a page. However, they are all more expensive to implement. We chose to monitor document selections because it might be a good enough indicator and it is much easier to implement than other measures. It is important to note, however, that the results in this study depend very heavily on this assumption.

The third hypothesis is that, given a large number of users that run the same query, the vast majority of them will tend to choose the same documents. We tested this hypothesis by building a search engine named Pluribus and making it available on the World Wide Web. The engine retrieves the results for queries it has never seen before by sending them to another Internet search engine, Metacrawler. It then returns the results to the user and monitors which documents she selects. When someone else types the same query the search engine combines the score assigned by Metacrawler to each document with the score derived from indirect relevance information, and then re-ranks the documents according to their new score. Documents that are more frequently chosen get an increment in their score computed by Metacrawler, and documents that are very rarely picked see a reduction in their score.

The following section describes Pluribus in more detail.

A brief description of Pluribus

Pluribus consists of a database that contains information about individual queries that have been submitted to the engine and a scoring mechanism that makes use of that information. Information about each query includes the list of documents that have been retrieved with standard IR techniques, their score, which of those documents have been selected in the context of the query, how many times, and how many times it should have been selected and was not. That is, the database contains a list of entries with the following information:

1. Query: The text of the query
2. Document: Information about a particular document that has been retrieved for the query
3. Score: The score of the document as computed with standard IR techniques (without the re-ranking mechanism)
4. Selections: The number of times that the document was selected by users in the context of the query
5. Expected Selections: The number of times that the document was expected to be selected by users but was not (also in the context of the query)

The database is initially empty. Pluribus does not attempt to index the World Wide Web, nor does it try to directly use Information Retrieval techniques to retrieve documents. Instead, Pluribus waits for a user to type a previously unseen query and then submits it to Metacrawler, a search engine that combines results from other engines (including Altavista, Excite, Lycos and Yahoo!). In order to increase the overlap in the queries it

receives, Pluribus pre-processes the original user queries by removing unnecessary differences. It first removes stop words such as *the*, *a*, *another*, etc., that will be ignored by Metacrawler anyway. It then re-orders the query terms in alphabetical order, another thing that is usually done by standard search engines[1]. The effect of this pre-processing is an increase in the overlap of queries in the database; it allows Pluribus to recognize that queries such as "*Agustin Schapira*" and "*Schapira Agustin*" refer to the same thing.

When Pluribus receives the list of pages that Metacrawler has retrieved for that query, it inserts all those pages in the database and then presents the results to the user. Each (query, document) pair has an associated score, as returned by Metacrawler; Metacrawler's scores go from zero to 1,000 points.

Once the results are submitted to the user, Pluribus monitors his/her actions and modifies its database accordingly. Every time the user selects a document from the results list, Pluribus records that selection and redirects the user's browser to the corresponding URL. For each selection, Pluribus increases the document's selection counter.

Pluribus acknowledges the fact that highly ranked documents have a higher chance of being selected than documents at the bottom of the list (because they have more visibility). Consequently, Pluribus puts more demand on highly-ranked documents; every time a document is selected, Pluribus goes through all the documents in the results list other than the one that has been selected and increments a value that indicates the expected number of times that this document should have been selected. The increment is a function of the ranking of the document: highly ranked documents get a high increment whereas documents in the bottom of the list get a very small increment. In the current implementation, Pluribus is set up so that it expects the top 25% pages in the list to be selected 60% of the time. Every time a user selects a document Pluribus increments the expected number of votes for top pages by (0.6 * the number of pages in the top group). The next 50% are expected to be selected 30% of the times (and thus the counter for the pages in the second group is incremented by 0.3 * the size of the group), and the pages at the bottom of the list are expected to be selected 10% of the times (and the counter is incremented 0.1 * the size of the last group). Note that, following this formula, the sum of all increments in expected selections that the system registers when it receives a document selection will always be equal to one. In other words, for each query the sum of expected number of selections across all its pages is always equal to the actual number of selections.

When Pluribus receives a query that is already in the database, it does not retrieve any new documents from Metacrawler but instead uses information it has already gathered and re-computes the score of each document.

The score of a document is computed as the combination of three elements:

[1] Some engines, such as Infoseek, are sensitive to re-arrangements of the query terms. Pluribus ignores that fact.

- The score assigned by Metacrawler. This score reflects the relevance of the page as computed by Metacrawler.
- 100 points for each selection that the page has received in the context of the current query.
- 100 points for each time that the page should have been selected.

Those three elements are combined in the scoring formula:

$$score = MC\_score + 100 * Selections - 100 * Expected\_Selections$$

Note that if a page is selected more times than expected then the difference between selections and expected selections is positive and thus the document gets a higher score than the one assigned by Metacrawler. Conversely, if the document is selected less often than expected, the difference is negative and its final score is less than the value assigned by Metacrawler. This mechanism thus is responsible for the re-rankings of the documents; as their score changes with time, documents occupy different positions in the list of results. Ideally, relevant documents will go to the top of the list and irrelevant ones will descend.

Also note that the value that is added to the counter of expected selections depends on the ranking of the document in the results lists, which in turn varies over time. Consequently, when a document is at the bottom of the list it gets a small penalty for not being selected. The fact that someone selects that document even though it is at the bottom of the list is a good indication that the document is relevant, and it is thus moved up the list very quickly. As more people select it and it approaches the top of the list, then the fact that someone selects it is less significant, which is reflected in an increased penalty and thus the speed of ascent slows.

Finally, it is important to point out that this scoring mechanism can become problematic as the number of document selections increases. As the documents in the results list for a query receive more and more selections, their original Metacrawler score will be superseded by the other elements of the scoring formula, and the IR part of each document's score will thus be ignored. This is not a desirable property, and a normalizing scheme has to be introduced in the future to balance that. In this experiment, however, the quantity of selections was never too large to become a problem (see Results section).

As has been described above, once Pluribus has cached the results for a query it always retrieves documents from its database. In order to avoid showing outdated information, Pluribus refreshes its cache once a week. When it receives a query that is stored in the database and whose contents are more than a week old, Pluribus submits it back to Metacrawler and maybe retrieves new documents that were not returned by Metacrawler the previous time. These new documents are inserted into the database, with their scores equal to the value returned by Metacrawler and with no information about number of selections or expected number of selections. One effect of this policy is that as time goes by, the size of the results list for a particular query increases. For example, Pluribus could receive 20 documents from Metacrawler the first time it submits the query. One week

later, Metacrawler might return two new documents that are inserted into Pluribus' database. A user that submits the query a week after the query was first received will get a results list with 22 documents, instead of 20 as the first user got.

Pluribus [7] can be accessed at http://keilor.cs.umass.edu/pluribus. Since it is an Open Source project, its code can be downloaded and modified for different experiments.

Figure 1: Pluribus

The following section analyzes the data obtained by monitoring the usage of Pluribus for two months.

Experiment Results

Pluribus remained open for public use for two months (April and May 1999), during which we recorded all the queries that were submitted, all the results that were presented, and all the selections that were made. We then used data from those logs to verify the hypothesis that repeated user selection is an indicator of document relevance.

In order to attract traffic, Pluribus was listed in web pages devoted to Internet search engines, announced in newsgroups and mailing lists, linked from the sites of the

platforms on which it had been developed (mySQL and PHP), and promoted in general search engines.

Pre-processing

Pluribus received 2,767 queries and a total of 1,683 document selections. Of those 2,767 queries 1,088 were unique, with each unique query submitted to Pluribus an average of 2.5 times.

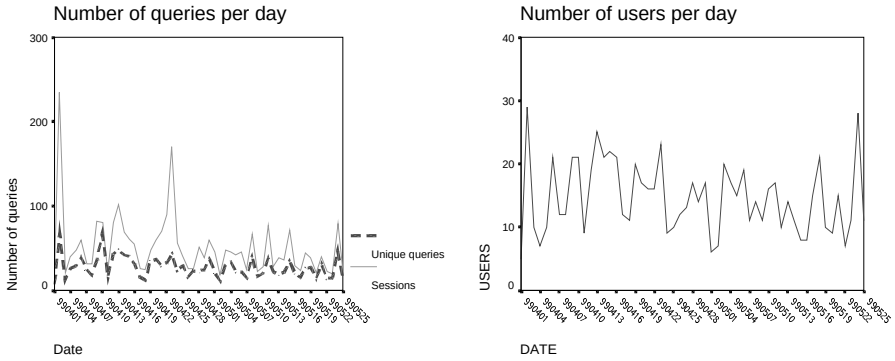


Figure 2: Pluribus usage

Of those 1,088 unique queries, 617 were submitted to Pluribus only once. Pluribus cannot improve the quality of the results for a query that it has processed only once, and we thus removed those 617 queries from the dataset for our analysis. (In a strict sense, Pluribus can start to learn from a single session, but there is no way to evaluate whether that learning will be meaningful in other circumstances)

In the case of many queries, users did not select any documents from the results lists. If there are no document selections then Pluribus cannot gather relevance feedback, and thus the system behaves exactly as the source from which it retrieves its results (Metacrawler in this case). We thus only kept in the dataset the 278 queries for which at least one document selection had occurred.

We then removed from our consideration all those queries that had been submitted only by a single person or for which selections had only been made by a single user. We identified users by the IP address of their machine or by the IP address of the proxy through which they accessed Pluribus. After this pre-processing step, only 31 queries remained for study.

We finally noticed that there were some queries that had been executed several times, by different users, and for which several selections had been made, but for which the selection ratio was very low. This implies that many users had executed the query but that the number of times that documents had been selected was comparatively very low. We set an arbitrary limit for selection rate (0.2, meaning that a document was selected at least once for every five times that query had been submitted) and were left with 18 queries to

study. Those 18 queries sum up to 509 different sessions and 368 document selections (a session consists of a query submitted by a user and all history of document selections performed).

The following table shows the chosen queries and their statistics.

Query	Submissions	# people that submitted it	# of selections	# people that made selections
sex	210	123	151	61
mysql	163	112	122	49
collaborative filtering	58	25	28	14
php	12	10	7	5
agustin schapira	11	6	6	3
php3	10	8	3	3
Pasantias	8	3	5	2
lv3 radio	5	2	4	2
mexicans sexy	5	3	4	3
"make your own cds"	4	2	4	2
dsp	4	3	3	2
Programming tutorial xlib	4	3	4	2
anderson pamela	3	3	4	3
empanada	3	3	7	2
emulator network	3	3	3	2
"sources of carbon dioxide"	2	2	8	2
Creatinina	2	2	3	2
e-machines	2	2	2	2
TOTAL	**509**	**315**	**368**	**161**

TABLE 1: Queries in the dataset

As can be seen from the previous table only three queries were executed more than 20 times or received more than 20 document selections, while the majority of the queries were only submitted two or three times. As a consequence, our data lend themselves to an anecdotal analysis of the performance of the system, but not to any exploration of statistical significance. In what follows we present an informal analysis of the data and provide anecdotal evidences suggesting that the approach could be valid for a certain type of queries.

Analysis

A first observation of the list of queries suggests the existence of two different groups. On one hand, there are popular queries, with more than 50 submissions each (*sex*, *mysql*, and *collaborative filtering*). On the other hand, the vast majority of the queries seem to be rare, with an average of three submissions/query.

Our analysis suggests that the performance of our approach might depend heavily on the *popularity* of the query to which it is applied. The data show that the system performs poorly for *popular* queries and better for *not-so-popular* ones. A possible explanation of

this behavior is related to how the number of selections determines the overlap in the documents that are selected by different users for the same query. If few people submit one particular query and they select a small number of documents from the results list (there were 1.17 selections per session in average), there is a high chance that they will all select documents from the same subset of *clearly relevant* documents. As more and more people submit the same query, however, there is a higher chance that some of them a) are looking for a different thing than the rest or b) are interested in exploring and selecting documents *other than the clearly relevant ones.* As different people explore other possibilities in the results lists then the overlap in documents selected becomes smaller, which in turn affects the performance of the system.

If this is the case, then a possible conclusion is that the system is trying to adapt too rapidly for popular queries, in which case selections should be taken with more reserve. In other words, the fact that a user selects a document for a rare query implies that the document has a high probability of being relevant; on the other hand, since so many people select documents for popular queries, then each selection should increase the system's expectation that the document is relevant by a smaller number. If this analysis is correct, then it could be possible to improve the performance of the system by adjusting the rate at which it re-ranks documents for popular queries.

We will now illustrate these observations with concrete data from Pluribus.

Popular queries

The most popular query is *sex*, with 151 selections in 91 different sessions (an average of 1.7 selections registered every time the query was submitted). Figure 3 shows the average ranking of the documents selected in each of the 91 sessions. This metric is computed by grouping all the document selections into their corresponding session and then computing their average ranking (rankings start from zero for the top document in the list). As was explained before, a session consists of the submission of a query by some user and all the documents that the user selects for that query in that submission; if the same user submits the same query later, that is taken as a different session. This metric helps analyze the performance of the system as time passes. If the scoring mechanism performs well we expect to see a decreasing trend in the average ranking of the selected documents; that behavior would be a consequence of good documents being selected and moving up in the list (and thus their ranking going down) and unpopular documents being demoted in the list (and thus their ranking going up).

Figure 3 shows that the system does not behave satisfactorily for the popular query *sex.* Instead of a decline in the average ranking of the documents selected, we observe a random pattern with regular peaks as the session number increases. In fact, on the right side of the graph we find an increase in the average ranking, a clear indication that the approach is not working properly.

A look at the list of documents that Pluribus returned for the query *sex* shows a wide range of topics. The titles and summaries of the documents retrieved were informative,

which implies that the users were making informed choices. However, it seems that different users were looking for different things when they typed the query *sex*, and that the wide variety of topics in the documents retrieved allowed each user to pick the particular topics they were interested in. This results in a lack of overlap in the documents selected, and ultimately in a poor performance of Pluribus.

A careful analysis of Figure 3 shows how the average ranking of the documents selected in each session depends heavily on the refresh periods of the system. As was explained in the previous sections, Pluribus refreshes the contents of its cache when they are older than a week. This refresh introduces new retrieved documents for each query, and as a consequence the number of documents returned to the users in the results list grows as time passes. The appearance of new documents (and thus the longer results lists) explain the peaks in the graph: after each refresh there are more documents to choose from. Some of the new documents are relevant but appear close to the bottom of the list, because they have not had a chance to be selected in the past and thus the only score they have is the one provided by Metacrawler (whereas the older documents have a Metacrawler score plus extra points for the selections they have received). Since some of these new documents at the bottom of the list are relevant, users tend to select them and thus the average increases. An ANOVA analysis of the groups of means within each refresh period shows that the averages increase significantly every time the system refreshes its cache.

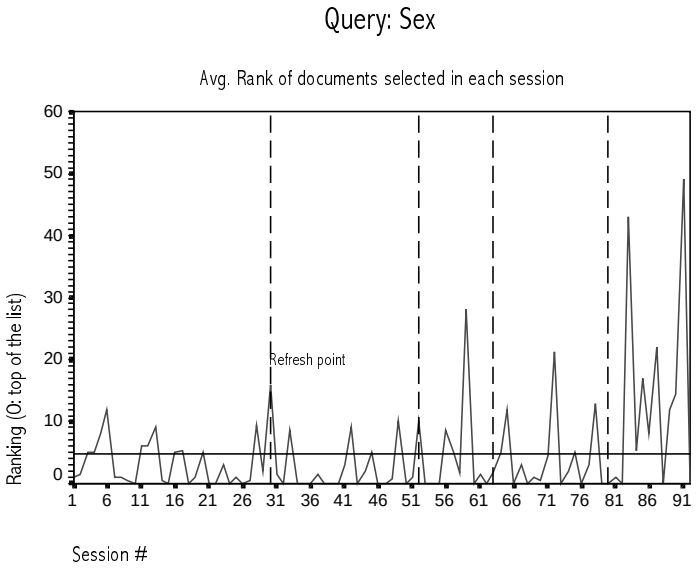


Figure 3: Query *sex*

A similar behavior is observed for another popular query, *mysql* (a database engine). The sequence graph for that query shows oscillations in the average ranking of the documents selected in each session. In addition, refresh points are followed by increments in that average, as newer documents appear at the bottom of the results list.

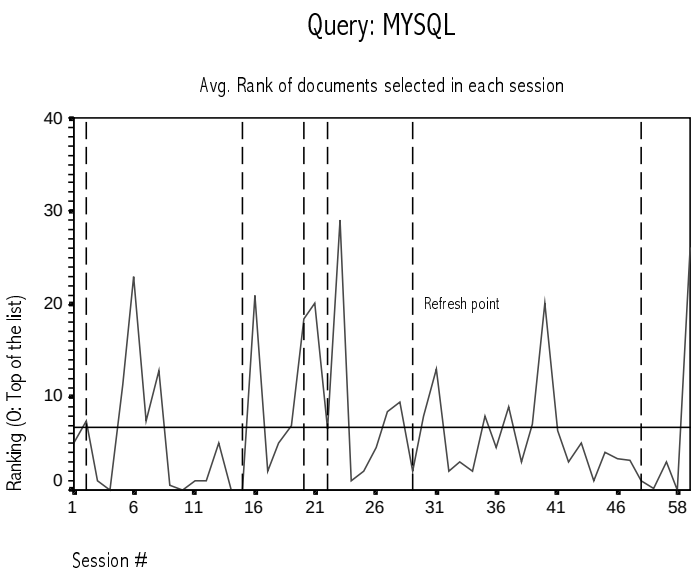


Figure 4: Query *mysql*

This analysis suggests two things:

- If the queries are very popular such as *sex*, then different users tend to select different sets of documents, and thus the system receives too much noise to be able to learn anything, and
- Refreshing the cache introduces the problem of increasing the number of documents that are shown to the users, increases the average ranking of selected documents because it introduces new (possibly relevant) documents at the bottom of the list, and makes comparisons more difficult.

The next query in popularity, *collaborative filtering*, received 28 selections in 16 session. In this query we can observe one characteristic present in the two other popular queries: increased averages after a refresh point.

This query, however, also serves as the transition point between the two groups, the frequent and the rare queries. Although the number of points is too small to make any solid conclusions, in the graph it seems that the system *is* learning to recognize the relevant documents and to take them to the top of the list. Within each portion of the graph between refresh points there is a clear decreasing line for the average of the documents selected. This implies that users tend to select documents at the top of the list as time goes by, until new (relevant) documents appear at the bottom of the list and the system has to learn again.

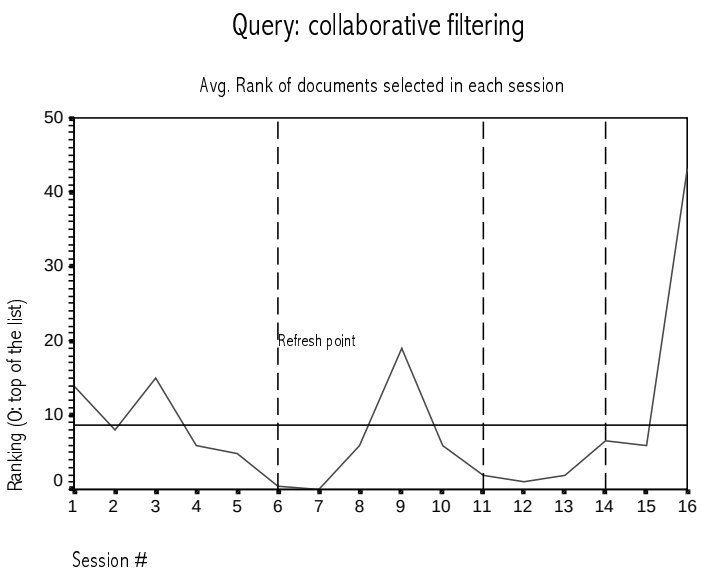


Figure 5: Query *collaborative filtering*

Less frequent queries

We had 15 queries in our dataset that were repeated often enough for the system to learn, but not as frequently as *sex* or *mysql*. In the majority of the cases (9 cases), the system was able to correctly identify relevant documents. In some cases (3) there was no overlap in the documents selected by different users and thus the system was unable to learn. In the rest of cases (3) the system did not improve its performance although there had been an overlap in the documents selected. The following table summarizes these results and the rest of the section examines each query in detail.

Result	Queries
Correct learning	*php, Agustin Schapira, php3, lv3 radio, mexicans sexy, programming tutorial xlib, Pamela Anderson, empanadas, e-machines*
No overlap in selections	*Make your own cds, emulator network, creatinina*
No learning	*Pasantias, Dsp, "sources of carbon dioxide"*

TABLE 2: Rare queries and system performance

In the queries where the system was able to learn, a decrease in the average ranking of the documents selected as time went by can be observed. It is important to emphasize the each query contains too few datapoints to reach any sound conclusions.

An interesting case is the query *php,* the most popular in this group (12 submissions); as is the case with the queries in the other group, cache refreshments introduce new documents and raise the average ranking of documents selected. Notice that Pluribus only refreshes the cache for an old query when it is re-submitted; if several weeks pass

between two submissions of the same query, then only one refresh is performed. This explains why popular queries have many more refresh points than rare ones.

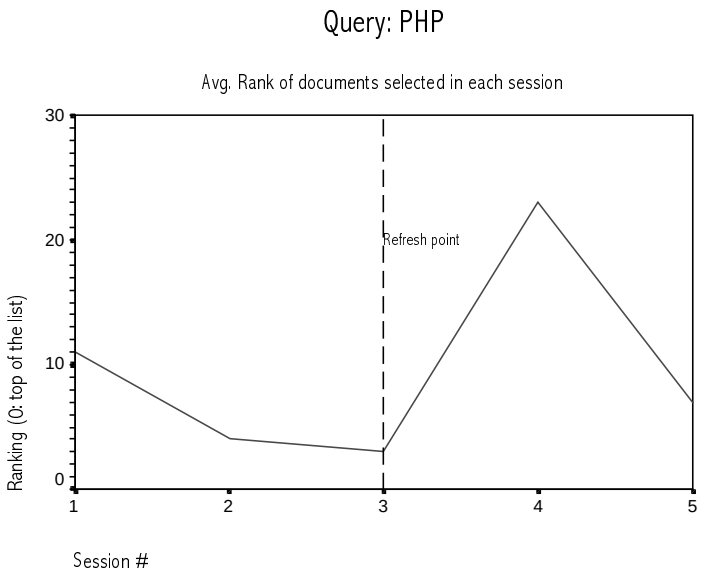


Figure 6: Query *php*

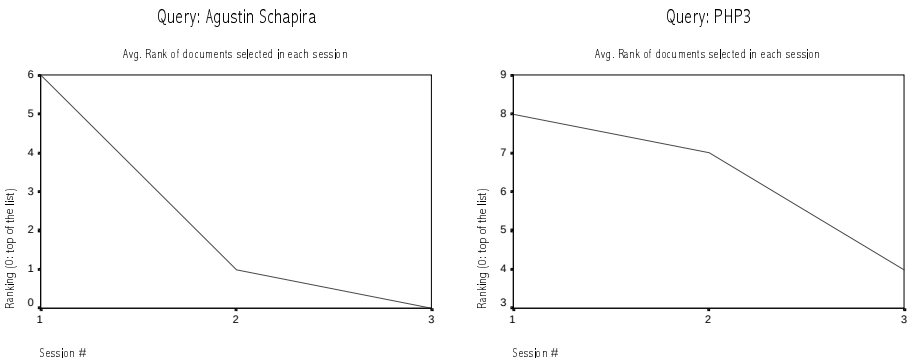
The query *Agustin Schapira* demonstrates an improvement due to the re-ranking mechanism[2]. The following is the sequence of selections registered:

Session #	Doc. Selected	At position	Original Position
1	40	4	4
1	626	2	2
1	627	3	3
1	637	15	15
2	40	1	4
3	162	0	0

QUERY: *Agustin Schapira*

The first user that submitted the query selected document #40 in the first place. That selection increased the document's score and moved it closer to the top of the list. When the next user submitted the query, he/she found it relevant and selected it from position 1, three points higher than its original position. In that sense, the system learned to identify the relevant document.

[2] The query with the author's name was submitted by real users of Pluribus, not by the author in a controlled experiment; those users were probably looking for more information about the author.



Figures 7 & 8: Queries *Agustin Schapira* and *php3*

A similar behavior can be observed for the *PHP3* query. The third user selected the same document than the first user chose; that document moved up four positions in the results list.

Session #	Doc. Selected	At position	Original Position
1	5525	8	8
2	13255	7	7
3	5525	4	8

QUERY: *PHP3*

The case of the query *pasantias* is very interesting and demonstrates one problem introduced by the use of Metacrawler to retrieve documents. When it receives a query, Metacrawler submits it to several search engines such as Altavista, Lycos and Excite and waits for their response. In order to avoid exposing the user to very long delays, Metacrawler times out if a search engine has not replied to the request. This generates cases in which most of the engines time out (probably because the Metacrawler server's connection to the Internet is experiencing problems) and thus the quality of the results is very poor. If the query is re-run at a later time and few or no engines time out, then the list of documents returned by Metacrawler will contain a large proportion of new good documents. This is what happened in the case of the *pasantias* query. The first time it was executed it returned very few good documents. When it expired and was refreshed a week later most of the old documents moved to the bottom of the list, since the new documents had a much higher score. This had a negative effect on Pluribus; the first user that ran the query chose one of the documents that were retrieved, and Pluribus moved it to the top of the list. When the query was refreshed, that document was moved to the 20th position because its score was very low compared to the score of the new documents. However, that document turned out to be relevant, and the next visitor selected it but this time at the bottom of the list; the average ranking thus increased notably.

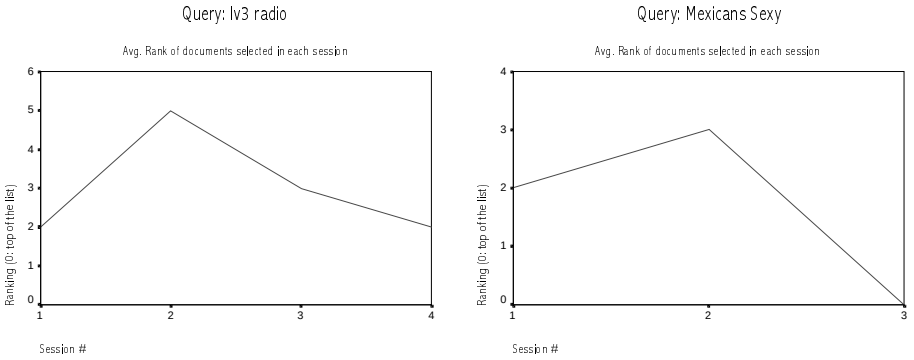
Session #	Doc. Selected	At position	Original Position
1	2943	2	2
2	14862	2	2
2	2950	24	9
2	2955	15	14
2	2943	**20**	3

QUERY: *pasantias*

Pluribus behaved very well for the queries "*lv3 radio"*, and "*mexicans sexy"*. In both cases, the average ranking of selected documents decreased as the system identified the relevant documents. The later query shows a very good behavior. According to the users' selections, document #4400 seems to be very relevant to the query: all the users selected it. Pluribus detected that and moved it from position two to position zero.

Session #	Doc. Selected	At position	Original Position
1	4400	2	2
2	4400	0	2
2	4404	6	4
3	4400	0	2

QUERY: *Mexicans sexy*



Figures 9 & 10: Queries *lv3 radio* and *mexicans sexy*

The query *programming tutorial xlib* is a case in which Pluribus moved a document up in the list not because it had been selected, *but because the documents above it were not selected enough times*. The following table shows how document 10797 was moved up eight positions because the documents above it had not been selected. This turned out to be a good move, because the second user selected it (at position 8 instead of 16).

Session #	Doc. Selected	At position	Original Position
1	10792	8	8
2	10789	5	5
2	10797	8	16
3	10786	2	1

QUERY: ***programming tutorial xlib***

Finally, the query *e-machines* also showed good results, although the number of cases is only two. Here, Pluribus moved a document from number 3 to number 0 after one selection; the next user also chose that document.

Session #	Doc. Selected	At position	Original Position
1	11348	3	3
2	11348	0	3

QUERY: ***e-machines***

No learning was observed in the other queries, either because there was no overlap in the documents selected or because there were not enough selections.

Conclusions and future work

This report presents anecdotal evidence that the proposed approach might be useful to improve the quality of the results returned by search engines in the case of queries of moderate frequency. It also indicates that the approach, as it was implemented in Pluribus, might not be valid in the case of very popular queries.

A plausible explanation of this difference is that in the case of popular queries the system increases or decreases documents' scores (and thus moves them) too rapidly. We have speculated that a possible solution to this problem might be to make the size of the changes in scores get smaller as the query becomes more popular. An experiment with that modified system will allow to analyze this explanation in detail and might help improve the performance of the system and make it valid for popular queries as well.

This preliminary experience has also been useful to delineate another future experiment to prove or disprove the original hypothesis with statistical significance. In the first place, it is clear from our experience that data has to be collected with a more popular search engine in order to get a clear picture of the real searching behavior in search engines. That picture is necessary to test the validity of our hypothesis. Thanks to its popularity, DirectHit has been able to collect a large amount of data about the searching behavior of its users, but regretfully they do not make that information public.

In future experiments it will be important to control the effect of Metacrawler's time-outs, which make it an unstable engine. In some cases Metacrawler cannot retrieve documents from some sources because the connection is down; submitting the query sometime later, when the problem has been fixed, returns a completely different set of

documents, usually with higher quality (and thus score). Introducing in Pluribus' database a new set of documents with much higher scores than the original set has the effect of erasing all the learning that the system has achieved in the past. In our preliminary experiment we used Metacrawler because it provided results from a wide selection of search engines; in more controlled experiments using a single stable engine such as Altavista is recommended.

Also, it will be important to control for the effect of refreshes of the cache; new documents introduce noise in the system. If ignoring new documents is not acceptable, then at least the system should always show only the top *n* documents of the list, although some documents that once appeared in the results list might disappear later. Keeping the size of the results list constant will also make it easier to compare the performance of the system as time goes by.

In this experience we have been able to identify a series of variables that can be used in future experiments to test the hypothesis that document popularity is a good indicator of relevance. If the hypothesis is true then we expect to find an improvement in the quality of results as users submit the same queries and select the same documents. Increased quality in this system means that relevant documents move to the top of the results list while non-relevant documents descend to the bottom. In principle, increased quality can be measured with the help of several variables:

- The average ranking of the documents selected in each session. For each particular query, we expect to see a decrease over time of the average ranking of the documents selected. This should be generated by the fact that relevant documents (and thus the ones users choose) tend to move higher on the list.
- The ranking of the first document selected by a user that submits a query. If relevant documents move to the top of the list as time goes by then users should tend to select documents at the top of the list first and only then move to less relevant one if they want more information.
- The number of documents that are selected from the top 5% of the list and the number of documents that are clicked from the bottom 5%. This measure is related to the previous and gives an idea of the precision of the documents at the top and the bottom of the list.
- The average ranking of the set of documents selected by users for each individual query. If the hypothesis is true and the system is working properly then the average ranking of the documents deemed *relevant* by the users should decrease.

If further experimentation supports these conclusions, then using the re-ranking mechanism proposed here could lead to a significant improvement in the quality of real Internet search engines even if it only works for not-so-popular queries. In those engines, many queries are executed very often and represent a very large percentage of all the queries, and many are executed only once or twice. In the middle of these two extremes there is a large number of queries that are less frequent than the most popular ones but

that yet are submitted by several different people. Our data suggest that the re-ranking mechanism proposed here could automatically generate better results for that large set of queries in the middle. For very popular queries, it might be more convenient to manually create a list of relevant documents, or improve the system so that it scales well to a large number of document selections. In the case of extremely infrequent ones only advances in IR techniques could lead to improved results.

References

[1] S. Lawrence and L. Giles. "Searching the World Wide Web". *Science Magazine* Volume 280, pp. 98-100.

[2] P. Resnik and H. Varian, guest editors. "Recommender Systems". *Communications of the ACM*, March 1997.

[3] M. B. Jansen, A. Spink, J. Bateman, and T. Saracevic. "Real life information retrieval: a study of user queries on the web". *SIGIR Forum*, 32(1), Spring 1998.

[4] J.J. Rocchio. "Relevance Feedback in Information Retrieval". In Gerald Salton, editor, "The SMART Retrieval System –Experiments in Automatic Document Processing", chapter 14. Prentice Hall, 1971.

[5] G. Salton and C. Buckey. "Improving retrieval performance by relevance feedback". *Journal of the American Society for Information Science*, 41(4):288-297, 1990.

[6] H. Lieberman, N. Van Dyke, and A. Vivacqua. Lets Browse: A Collaborative Browsing Agent . *International Conference on Intelligent User Interfaces*, January 1999.

[7] Pluribus. http://keilor.cs.umass.edu/pluribus . Pluribus can be downloaded from http://keilor.cs.umass.edu/pluribus/distr/ .

[8] DirectHit. http://www.directhit.com .