Optimizing Ranking Calculation in Web Search Engines: a Case Study

Miguel Costa[1] and Mário J. Silva[1]

[1]Departamento de Informática
Faculdade de Ciências da Universidade de Lisboa
1749-016 Lisboa, Portugal

mcosta@xldb.fc.ul.pt, mjs@di.fc.ul.pt

Abstract

Web search engines compete to offer the fastest responses with highest relevance. However, as Web collections grow, it becomes more difficult to achieve this purpose. As most users tend to see only the first two pages of results, it is unnecessary to compute the ranking of each one of the millions of documents that usually match any given query. Only those that have a level of importance that makes them candidates to the top ranked results have to be considered. This work presents and compares algorithms tested in our Web search engine to speed up the search of these candidates. We have been able to reduce by 93% the number of documents considered for ranking calculation, using a pruning algorithm over a Web collection index sorted by URL weights.

1. Introduction

The size of collections indexed by some of the major Web search engines reaches today more than 3 billion pages. A search result (list of documents that match a query) can have more than a hundred million pages. If search engines had to calculate the ranking for all these documents, response times would be unacceptable.

Fortunately, most users don't see all the results returned for their searches, but only the first ones. If the documents are sorted in the index by a measure of importance independent of the query, it is possible to restrict the ranking calculation to a subset of the candidate documents containing the documents that users are likely to see. This approach is followed by major search engines to limit response times. Google used to rank a maximum of 40,000 document matches for each query, offering sub-optimal results [1].

In the development of tumba! [21], a Web search engine for the Portuguese Web, we analyzed techniques to filter irrelevant documents from ranking calculation. We researched how many documents have to be ranked to produce results without loss of quality and how much processing can be saved. Solutions to this problem are necessary now and will be even more in the future, since Web collections tend to grow.

This paper is organized as follows. In Section 2, we cover work related to the problem of reducing ranking calculation, while in Section 3 we formalize that problem. In Section 4 we describe the path for our solution. Results are presented in Section 5 and analyzed in Section 6. Section 7 presents the conclusions.

2. Related work

The filtering of irrelevant matches before ranking query results has been widely studied for decades in many querying scenarios. Initially, ranking algorithms were based on statistics

about the textual content of the documents and collections [20]. The first developed pruning algorithms were based on these statistics. As other data began to be used in ranking calculation, specially Web linkage, pruning algorithms also evolved to account for this new information.

Independently of the information used, pruning algorithms may be classified as either safe or unsafe. Safe algorithms reduce computation without affecting results [3, 18, 5, 15, 7, 2]. Unsafe algorithms trade quality of results for speed, by relaxing mathematical guaranties of the results' correctness [14, 4, 13]. Both classes of algorithms operate in general over inverted indexes, the most used and best performing index structure in search engines [28].

From the wide diversity of pruning algorithms described in the scientific literature, we summarize the ones from which we extracted ideas for our work. The upperbound search algorithm developed by Buckley and Lewit contains the main idea of safe algorithms for ranking optimization [3]. First, it orders the query terms by their weight in decreasing order. Then, it iteratively processes the similarity between each query term and the matching documents, until the computed upper bound of the $k + 1$th document with higher similarity is smaller than the similarity of the kth. This algorithm mathematically guaranties accurate results for the top k documents. However, improvement isn't significant.

Techniques to filter terms and their associated posting lists from ranking calculation cause a significant degradation in retrieval effectiveness. Persin developed a technique that filters the documents of the posting lists individually [18]. The decision of whether to process or reject a document depends only on the number of occurrences of a term in the document, tf. Thus, all the documents in the posting lists are sorted by their tf, enabling the processing of the most important documents first. This technique achieved a 80% reduction of search times without degrading the results of the system used by the proponents.

Today, ranking algorithms use more than one type of information. To reduce ranking calculation under these conditions, several inverted files could be created, one for each type of information. Safe "top-k" algorithms, such as Fagin's algorithm or the Threshold algorithm (TA), can be used to find the k documents more relevant for a query [5, 15, 7]. The TA algorithm does sorted access to posting lists pre-sorted by their ranking values. First, all the top values of the inverted files are accessed, then the seconds, and so on. After k candidate documents have been accessed, a threshold t is defined as the smallest ranking value of the candidates. It continues calculating the ranking value of the documents until it gets one value inferior to t.

This algorithm presents two drawbacks. First, we must have as many indexes as the ranking criteria used. Second, some ranking metrics, such as the proximity between query terms on a text, can't be pre-sorted. TA-Adapt, is a TA variant developed by Bruno et al. that uses only one sorted index to access values sequentially and the other indexes randomly [2]. Let the threshold t be the lowest ranking between all the documents in the candidates set, and $U(d)$ the upper limit that a ranking value of a document d can have. $U(d)$ is computed with the known ranking value from the sorted list, and the remaining ranking values are set to the maximum value they can have. TA-Adapt first gets the top k documents from the sorted inverted list, called the candidates set, and computes their ranking probing the other ranking values from other sources. Then, TA-Adapt repeatedly accesses the next document d from the sorted inverted list until $U(d) < t$. When this happens, it stops and returns the top k documents from the candidates. Until then, if $U(d) \geq t$, the ranking of d is computed with all the probed ranking values and compared against t. If the ranking value is superior to t, the document is added to candidates and t is recomputed, or ignored otherwise. There are also variants of TA-Adapt that use the knowledge of the ranking function to first probe the attributes that have more weight and decide

faster if the documents' rankings are inferior than the threshold [2].

3. Reducing ranking calculation

Search engines results are produced in two steps. First, they match the documents that satisfy a query q. We call these the *query matches for q*, denoted as $QMatches_q$. Then, search engines rank each document d in $QMatches_q$, applying a ranking function $rank$ to produce a list of the query results for the user, denoted as $[QMatches_q]_{rank}$.

Evidence shows that on most queries, users only see the top k documents of $[QMatches_q]_{rank}$, represented as $[QMatches_q]_{rank}^k$. Therefore, it isn't necessary to compute the ranking score for all the documents in $QMatches_q$. The ranking function of a query q, $rank$, is composed of two sub-functions:

$$rank(d, q) = c * imp(d) + (1 - c) * sim(d|q)$$

$imp(d)$ weights the *importance* of each document d in the collection independently of the query. PageRank is an example of an importance function [16, 8]. $sim(d|q)$ weights the *similarity* between a document d and a query q. Algorithm $tf \times idf$ is an example of a similarity function [20]. Both functions can be composed by other functions of the same type. c is a coefficient to balance the weight of functions.

Our technique to reduce the ranking calculation, is based on filtering those documents d whose importance $imp(d)$ is not large enough to rank them among the most relevant, independently of the query q. First, all $imp(d)$ scores are computed for each document d in the collection, and the posting lists sorted by these scores. As the function imp is independent of the query, this processing can and should be performed offline.

When a query is processed, the top n documents in $[QMatches_q]_{imp}$ are selected, denoted as $[QMatches_q]_{imp}^n$. The value of n should be sufficiently large to contain all the documents that would be seen by the user if all matching documents were ranked, but as small as possible to reduce maximally the number of documents to compute the ranking. If n is large enough for the subset $[QMatches_q]_{imp}^n$ to contain all documents of subset $[QMatches_q]_{rank}^k$, after applying the function $rank$ to both subsets, the rankings produced will be identical for the top k documents. So, it is necessary to find a n such that:

$$\left[[QMatches_q]_{imp}^n \right]_{rank}^k = [QMatches_q]_{rank}^k$$

After finding n, it is only required to compute online the similarity between query q and the top n documents that match q, to obtain the same top k ranked documents for q. To find n, is necessary a $\tilde{n}$ function that computes for any query q the value of n.

Dissected the problem, our solution needs to find the best values for the variables involved in the problem of reducing ranking calculation. That is, for a set of queries Q with size $|Q|$, find the highest reduction in ranking calculation evaluated with the $reduction$ function:

$$reduction(k, \tilde{n}) = 1 - \frac{\sum_{i=1}^{|Q|} \tilde{n}(|QMatches_i|, k)}{\sum_{i=1}^{|Q|} |QMatches_i|}$$

4. Searching for a good solution

As the imp and sim functions, and the constant c in the rank function are arbitrary, it is impossible to find a universal solution to the problem independent of these functions. This

year		1998/2000	1998	2002	2002	2003
search engine		Excite	Altavista	Excite	Fast	Tumba!
queries		51473	1 billion	1025910	451551	356629
pages viewed	**avg**	2.35	1.39	1.7	2.2	1.46
	1	58%	85.2%	?	?	78.9%
	2	19%	7.5%	?	?	9.6%
	3	9%	3.0%	?	?	4.7%

Table 1: Result pages seen by users in Web search engines.

section presents ranges and alternatives for the various parameters of the $reduction$ function, based on the data collected from our search engine.

4.1. Choosing k

In most cases, users only see a small fraction of all the results that match a query. In a typical interaction between a user and a search engine, users make queries to search engines and receive a list of linked results, normally 10. These results are ranked by their relevance to the query.

Jansen et al. analyzed in 1998 (and again in 2000) 51473 queries of the query log of the Excite search engine [10, 11]. Silverstein et al. analyzed in 1998 approximately 1 billion queries collected over 43 days from the query log of the Altavista search engine [22]. Jansen et al., conducted another study in 2002 where they evaluated and compared the Excite and AllTheWeb search engines [23]. We also performed our own analysis of a subset of the tumba!'s log with 356629 queries. The collected results are summarized in Table 1. They indicate that the majority of users tend to browse only the first two pages of results (top 20 results), sometimes browsing the third result page.

k should have a value high enough to cover the results that most users usually see. Given the data presented above, we evaluated the $reduction$ for $k = 10$, 20 and 30. Considering a value of k higher than 30 (3 result pages with 10 results), would produce a negligible difference in observed results for the vast majority of queries.

4.2. Choosing ranking functions

4.2.1. imp functions. To evaluate the effect of imp in the reduction of calculations, we chose 2 functions. One is a variation of PageRank, which we call extPageRank. PageRank computes an importance value for each page using the Web graph with an equal and full flow in all the links. Our variation assigns a 10% flow to internal links, since most of them are navigational and do not correspond to an importance given by independent authors. This percentage was tuned empirically.

The distribution of extPageRank follows a power law distribution (see Figure 1). This is similar to the PageRank distribution [17, 9, 25]. The higher extPageRank values are very sparse and differences among them do no reflect the disparity of importance. We segmented the extPageRank values and assigned a weight to each of these segments (see Table 2). This seems to be what Google does, based on the observed normalized scores between 0 and 10

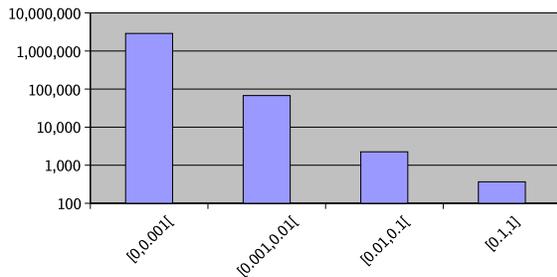| imp functions		weight
extPageRank	URL weighting	
[0,0.001]	file	0
]0.001,0.01]	path	0.25
]0.01,0.1]	subroot	0.5
]0.1,1]	root	1

Table 2: Weights given to each class.

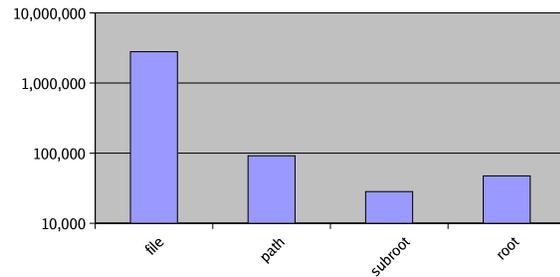Figure 1: extPageRank distribution.

Figure 2: URL weight distribution.

shown on Google toolbar [25].

The second imp function is computed as the URL weighting algorithm that achieved the best results on the TREC-2001 home page finding task [27, 12]. It is based on the observation that documents at the root URL of a specific site are often entry pages. As we descend deeper in the site's directory tree, the probability of being an entry page seems to be inversely proportional. In our algorithm, the URLs are normalized and then divided in four types:

root: a domain name, e.g. www.tumba.pt
subroot: a domain name followed by a single directory, e.g. www.tumba.pt/pt/
path: a domain name followed by more than one directory, e.g. www.tumba.pt/pt/help/
file: anything ending in a filename, e.g www.tumba.pt/pt/about.html

The URL class distribution (see Figure 2) also has similarities with other works [27, 12, 26]. The difference is that our crawl of the Portuguese Web has more pages belonging to the root class than to the subroot class. We conjecture that this results from having used as seeds, a list of registered Web domains whose Web sites mostly have only a homepage. Each class was weighted as shown in Table 2.

The two distributions have correlation of 0.477.

4.2.2. sim **functions.** Our sim function adds similarities between query terms and documents, using different weights for special sections of the documents where the terms occur (e.g. in URLs, titles, anchors). The scores produced by the sim function are normalized between 0 and 1.

4.2.3. coefficient c**.** The coefficient c that balances the relative weight of the imp and sim functions, was set in this study to values of 0.5, 0.66 and 0.8, corresponding to weight ratios of $\frac{c}{1-c} = 1, 2$ and 4.

4.3. Determining $\tilde{n}$

The function $\tilde{n}$ represents the distribution of n for a query set Q. As this distribution should contain n values as small as possible to reduce ranking calculation, pruning algorithms were used to compute these n values over the inverted index sorted by a imp function.

In the optimization of the tumba! ranking algorithm we studied two representative algorithms, one safe and another unsafe. As safe algorithm, we use TA-Adapt on posting lists pre-sorted by imp. TA-Adapt guaranties that the results of the top k documents will be produced as if the ranking was computed for all the documents. We denote as *safe/TA-Adapt* the $\tilde{n}$ function representing the distribution of the TA-Adapt algorithm.

As unsafe algorithm, we studied one based on the analysis of tumba!'s statistical data on queries and documents, denoted Stat. This algorithm first, computes for each query the ranking for all matching documents and gets the top k results. Then, it fetches the documents by the imp sorted order of the inverted index posting lists, until collecting the same k documents. The number of fetched documents is the n value. Computing a linear regression function using these n values, we get a function $\tilde{n}$ that statistically predicts n for any query. We denoted it as *unsafe/Stat* $\simeq n$. However, as this function doesn't completely cover all n documents, we also computed a linear function $\tilde{n}$ that covers all n documents and offers the highest possible reduction. Its coefficient was computed as the highest value of $\frac{n}{|QMatches_q|}$ for all queries. We denoted it as *unsafe/Stat* $\geq n$.

5. Results

We observed the reductions over a Web collection composed by 3,235,140 Web documents (see [6] for a detailed characterization of this collection). For the 100 most frequent queries submitted to our Web search engine during a one year period, we analyzed the distribution of n for the *unsafe/Stat* $\simeq n$, *unsafe/Stat* $\geq n$ and *safe/TA-Adapt* functions, with the extPageRank and the URL weighting algorithms as imp functions. Results were produced for the three k values considered: 10, 20 and 30, always using a coefficient c equal to 0.5. At the end of this section, we observe how reductions change as we vary c.

5.1. *unsafe/Stat* $\simeq n$ function

5.1.1. imp as extPageRank. Using the Stat algorithm over the index sorted by extPageRank, we computed the n values for each of the queries. As the graphics in Figure 3 shows, n increases almost linearly with $|QMatches_q|$. By applying linear regression to n values, we defined a function $\tilde{n}$ which predicts n for each query, parameterized with $|QMatches_q|$ and k. This function is depicted as the continuous line in the graphics of Figure 3. For the three different values of k considered, we computed $\tilde{n}$. As k increases, n increases a bit. This increase is expected, since more documents of the subset $[QMatches_q]_{imp}^n$ become necessary to ensure it contains the additional documents in $[QMatches_q]_{rank}^k$.

All graphics show that n is smaller than $|QMatches_q|$. Therefore, it is possible to achieve a reduction in ranking calculation. Figure 7 plots the reductions achieved in ranking calculation. The number of documents ranked was reduced by 24.13% for a k of 10, 21.35% for a k of 20, and 20.72% for a k of 30.
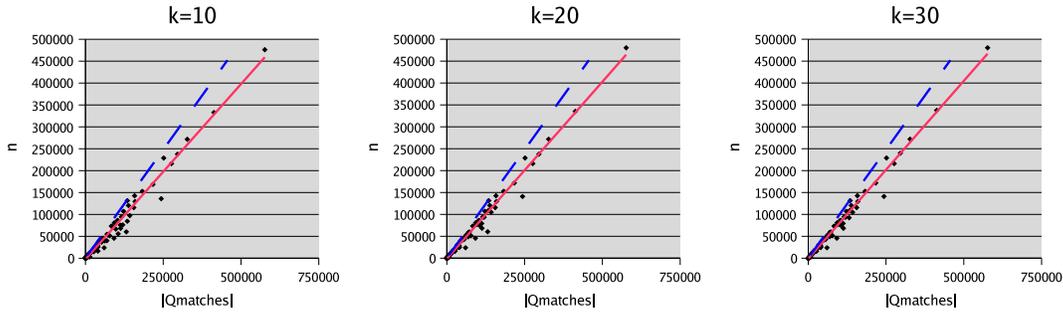
Figure 3: $\tilde{n}$ as a function of n using Stat over the index sorted by extPageRank.
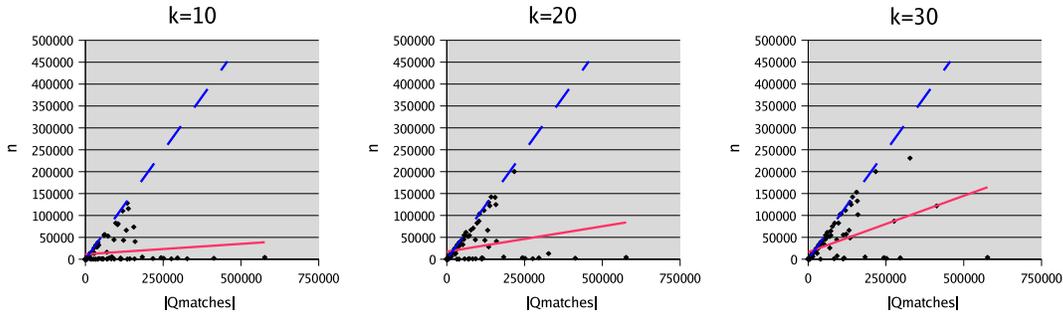

Figure 4: $\tilde{n}$ as a function of n using Stat over the index sorted by URL weight.

5.1.2. imp **as URL weighting.** Doing the same tests with Stat over the Web collection index sorted by URL weight, we got the distribution of n presented in Figure 4 for the different k values. Unlike extPageRank, this algorithm doesn't present a linear distribution for n and consequently, the function produced using the linear regression of these points can't predict much.

5.2. *unsafe/Stat $\geq n$* **function**

Independently of the imp function used, $\frac{n}{|QMatches_q|} \simeq 1$ for all n values, so almost all the documents must be ranked to get optimal results (depicted as the dashed line in the graphics of Figure 3 and Figure 4).

5.3. *safe/TA-Adapt* **function**

5.3.1. imp **as extPageRank.** Using the TA-Adapt algorithm over the index sorted by the extPageRank algorithm, we get a slightly lower reduction than the one achieved with Stat (see Figure 5). The reduction is always 18.33% for the three k values, as depicted in Figure 7. The justification is that for each query, the top n documents ranked with extPageRank have an identical imp weight.

5.3.2. imp **as URL weighting.** Using the TA-adapt algorithm over the index sorted by URL weight, we achieved very good results (see Figure 6). For the top 10 to 30 results we get a
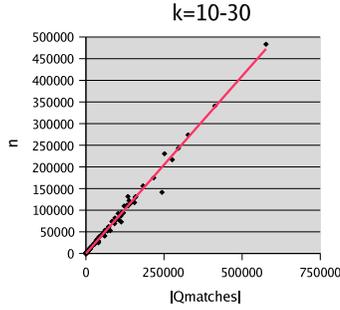
Figure 5: $\tilde{n}$ as a function of n using TA-Adapt over the index sorted by extPageRank.
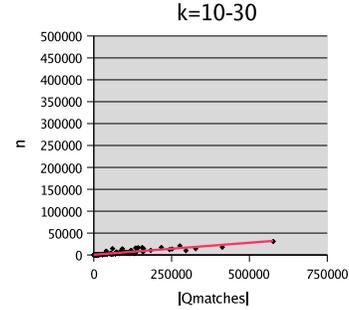
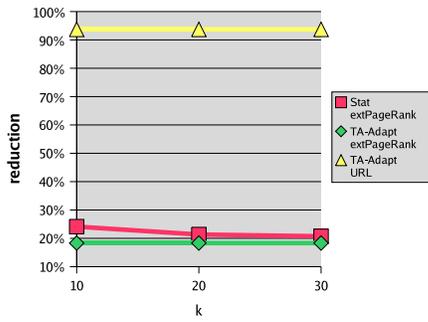Figure 6: $\tilde{n}$ as a function of n using TA-Adapt over the index sorted by URL weight.

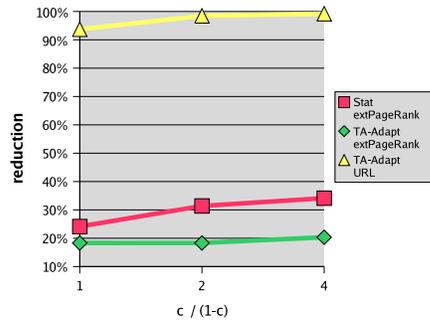Figure 7: Reductions using several algorithm combinations.

Figure 8: Reductions using several values for the c coefficient in the $rank$ function.

93.71% reduction (see Figure 7).

5.4. varying coefficient c

In previous results the coefficient c had always the value of 0.5 ($\frac{c}{1-c} = 1$) to balance the contributions of the imp and sim functions to the final rank. To measure the impact of this parameter in observed results, we conducted the same tests varying the coefficient c to 0.66 and 0.8, corresponding to $\frac{c}{1-c} = 2$ and 4. Results are presented in Figure 8 for a $k = 10$.

The reduction increased for all algorithms. As the weight of imp increases, the similarity between the rankings of the imp and $rank$ functions also increases. For $\frac{c}{1-c} = 4$, the algorithms Stat using extPageRank, TA-Adapt using extPageRank and TA-Adapt using URL weighting, achieved a reduction of ranking calculation of 34.13%, 20.35% and 99.18%, respectively.

6. Results analysis

The *unsafe/Stat* $\simeq n$ function can only be applied to reduce ranking calculation, when the index is sorted by the extPageRank algorithm. The distribution of n is close to its linear regression, so it can statistically predict a value n for all queries. It reduced ranking calculation by 24.13% when only the top 10 results needed to be exact. When the index is sorted by the URL weighting algorithm, n becomes very difficult to predict.

The *unsafe/Stat* $\geq n$ function, which covers statistically all n values, must rank almost all

the documents to get optimal results. This makes this function a bad choice, independently of the imp function used.

Using the *safe/TA-Adapt* function, we guarantee that $\tilde{n}$ covers mathematically all the n documents and consequently all the top k results are identical as the ones produced if the ranking was calculated for all the matching documents. Using it with the extPageRank algorithm to sort the index, we reduced by 18.33% the ranking calculation. With the URL weighting algorithm, the reduction raised to 93.71%. This large increase results from differences in the distributions of the extPageRank and URL weighting algorithms (see Figure 1 and Figure 2). extPageRank has much less documents in the classes with higher weight than the URL weighting algorithm. Consequently, algorithms with a power law distribution, such as extPageRank, tend to return most of the documents with the same imp value, the one assigned to lower classes. It become then difficult to distinguish the more important documents from the less important. Thus, safe algorithms must evaluate many documents before finding a document which mathematically guaranties that it is safe to prune the rest of the documents. Stat presents better results for this class of algorithms. On the other hand, algorithms with a more uniform distribution, such as the URL weighting algorithm, have better results when combined with safe algorithms. This uniform distribution spreads much more documents in the higher classes, which tend to fill the first positions of the posting lists, enabling to quickly select the most important documents for ranking calculation.

There is another important perspective to compare these algorithms. Web search engines of today index up to billions of documents and their indexes need to be partitioned by clusters of machines. This partition can be local, that is, each machine has an inverted index of a disjoint set of documents of the collection, or global, that is, each machine has an inverted index with a disjoint set of terms of the collection [24, 19]. With the local partition, all machines rank their subset of documents. The results from all the machines are then merged and the top k replied to the user. With the global partition, sets of documents are repeatedly requested from the machines with query terms indexed, and joined as necessary until having the documents ranked and the top k replied.

The algorithms analyzed, can be applied to Web search engines independently of the partitioning used. Beyond reducing computation, they enable a reduction of the number of postings fetched on disk. They also enable to reduce the number of messages requesting more documents when using global partition. Safe algorithms need to request sets of documents as necessary, until having the n documents that guaranty the top k documents ranked. We recall that safe algorithms only know n after evaluating n documents. The statistical algorithm used, Stat, can predict n without previously requesting any document. Therefore, combining safe algorithms with Stat, we get the best of two worlds: (1) safe algorithms warrant that results are exactly the same as ranking all matching documents; (2) offer the best reduction if the index is sorted by a ranking algorithm with uniform distribution; (3) Stat can compute a statistical function $\tilde{n}$ for a safe algorithm (e.g. in Figures 5 and 6), to predict n apriori and minimize the number of requests.

7. Conclusion

As Web users and Web collections tend to grow, it becomes more difficult for Web search engines to offer fast results with good quality. As users of Web search engines tend to see on average only the first two pages of results, it is unnecessary to calculate the ranking for all the

documents that match a query. Only a subset of candidate documents which contain the documents that users usually see need to be ranked. Our evaluation of combinations of algorithms to filter documents considered irrelevant for ranking calculation, lead us to the conclusion that the TA-Adapt algorithm over a Web collection index sorted by URL weights, reduces by 93% the number of documents to rank with no difference in the results seen by end users.

We also developed a statistical algorithm which can accurately predict the number of postings that have to be fetched from disk in a single i/o transfer. When using a global partition of indexes on distributed architectures, as we do in our Web search engine, this brings significant reduction on the number of exchanged messages during query processing.

References

[1] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[2] Nicolas Bruno, Luis Gravano, and Amelie Marian. Evaluating top-k queries over Web-accessible databases. In *Proceedings of the 18th International Conference on Data Engineering*, April 2002.

[3] Chris Buckley and Alan F. Lewit. Optimization of inverted vector searches. In *Proceedings of the 8th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 97 – 110, 1985.

[4] David Carmel, Doron Cohen, Ronald Fagin, Eitan Farchi, Michael Herscovici, Yoëlle S. Maarek, and Aya Soffer. Static index pruning for information retrieval systems. In *Proceedings of the 24th ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 43–50, 2001.

[5] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 102–113, May 2001.

[6] Daniel Gomes and Mário J. Silva. Characterizing a national community Web (accepted for publication). *ACM Transactions on Internet Technology*, 5(2), May 2005.

[7] Ulrich Guntzer, Wolf-Tilo Balke, and Werner Kiessling. Optimizing multi-feature queries for image databases. In *Proceedings of the 26th International Conference on Very Large Databases*, pages 419–428, September 2000.

[8] Taher H. Haveliwala. Efficient computation of PageRank. Technical report, Stanford University, 1999.

[9] Taher H. Haveliwala. Efficient encodings for document ranking vectors. Technical report, Stanford University, December 2002.

[10] Bernard J. Jansen, Amanda Spink, Judy Bateman, and Tefko Saracevic. Searchers, the subjects they search, and sufficiency: A study of a large sample of Excite searches. In *Proceedings of WebNet 98 Conference*, 1999.

[11] Bernard J. Jansen, Amanda Spink, and Tefko Saracevic. Real life, real users, and real needs: a study and analysis of user queries on the Web. *Information Processing and Management*, 36(2):207–227, 2000.

[12] W. Kraaij, T. Westerveld, and D. Hiemstra. The importance of prior probabilities for entry page search. In *Proceedings of the 25th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 27–34, 2002.

[13] Xiaohui Long and Torsten Suel. Optimized query execution in large search engines with global page ordering. In *Proceedings of the 29th International Conference on Very Large Databases*, pages 129–140, September 2003.

[14] Alistair Moffat and Justin Zobel. Fast ranking in limited space. In *Proceedings of the 10th International Conference on Data Engineering*, pages 428–437, February 1994.

[15] Surya Nepal and M. V. Ramakrishna. Query processing issues in image (multimedia) databases. In *Proceedings of the 15th International Conference on Data Engineering*, pages 22–29, March 1999.

[16] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: Bringing order to the Web. Technical report, Stanford Digital Library Technologies Project, 1998.

[17] Gopal Pandurangan, Prabhakara Raghavan, and Eli Upfal. Using PageRank to characterize Web structure. In *Proceedings of the 8th International Conference on Computing and Combinatorics*, pages 330 – 339, 2002.

[18] Michael Persin. Document filtering for fast ranking. In *Proceedings of the 17th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 339–348, August 1994.

[19] Berthier A. Ribeiro-Neto and Ramurti A. Barbosa. Query performance for tightly coupled distributed digital libraries. In *Proceedings of the 3rd ACM Conference on Digital libraries*, pages 182–190, June 1998.

[20] G. Salton. *Introduction to Modern Information Retrieval*. McGraw-Hill Computer Science Series, New York, 1983.

[21] Mário J. Silva. The case for a Portuguese Web search engine. In *Proceedings of the IADIS WWW/Internet 2003 Conference*, November 2003.

[22] Craig Silverstein, Monika Henzinger, Hannes Marais, and Michael Moricz. Analysis of a very large Altavista query log. Technical Report 1998-014, Digital Systems Research Center, 1998.

[23] Amanda Spink, Seda Ozmutlu, Huseyin C. Ozmutlu, and Bernard J. Jansen. U.S. versus European Web searching trends. *SIGIR Forum*, 36(2):32–38, 2002.

[24] Anthony Tomasic and Hector Garcia-Molina. Performance of inverted indices in distributed text document retrieval systems. In *Proceedings of the 2nd International Conference on Parallel and Distributed Information Systems*, pages 8–17, 1993.

[25] Trystan Upstill, Nick Craswell, and David Hawking. Predicting fame and fortune: PageRank or Indegree? In *Proceeding of the 8th Australasian Document Computing Symposium*, December 2003.

[26] Trystan Upstill, Nick Craswell, and David Hawking. Query-independent evidence in home page finding. In *Proceeding of the ACM Transactions on Information Systems (TOIS) archive*, volume 21, July 2003.

[27] Thijs Westerveld, Wessel Kraaij, and Djoerd Hiemstra. Retrieving Web pages using content, links, urls and anchors. In *TREC-2001 Notebook Proceedings*, 2001.

[28] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes: Compressing and indexing documents and images*. Morgan Kaufmann, 1994.