Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 48 (2015) 210 – 216

Procedia Computer Science

International Conference on Intelligent Computing, Communication & Convergence

(ICCC-2015)

Conference Organized by Interscience Institute of Management and Technology,

Bhubaneswar, Odisha, India

Web Crawling through Shark-Search using PageRank

Jay Prakash[a], Rakesh Kumar[b]
[a,b]Department of Computer Science & Engineering
Madan Mohan Malviya University of Technology
Gorakhpur, 273010, U.P. , India
{**Jpr_1998@yahoo.co.in**, **rkiitr@gmail.com**}

Abstract

This paper introduces the Shark search algorithm using PageRank algorithm, a refined version of one of the first dynamic Web search algorithms, the "PageRank search" and Shark-search. The Shark-search has been embodied into a dynamic Web site mapping that enables users to tailor Web maps to their interests. Preliminary experiments show significant improvements over the original page-rank algorithm**.**

Keywords: PageRank, Web Crawling, Shark-Search ;

1. INTRODUCTION

A crawler is a program that automatically collects Web pages to create a local index and/or a local collection of web pages. Roughly, a crawler starts with an initial set of URLs, called seed URLs. It first retrieves the pages identified by the seed URLs, extracts any URLs in the pages, and adds the new URLs to a queue of URLs to be scanned. Then the crawler gets URLs from the queue (in some order), and repeats the process. Every page that is scanned is given to a client that saves the pages, creates an index for the pages, or summarizes or analyzes the content of the pages [1-4].

The rapidly growing web graph contains several billion nodes, making graph-based computations very expensive. One of the best-known web-graph computations is PageRank, an algorithm for determining the "importance" of Web pages. The core of the PageRank algorithm involves repeatedly iterating over the web graph structure until a stable assignment of page-importance estimates is obtained. As this computation can take several days on web graphs off several billion nodes, the development of techniques for reducing the computational costs of the algorithm becomes necessary for two reasons. Firstly, speeding up this computation is part of the general efficiencies needed for improving the freshness of a web index. Secondly, recent approaches to personalize and topic

1877-0509 © 2015 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).
Peer-review under responsibility of scientific committee of International Conference on Computer, Communication and Convergence (ICCC 2015)
doi:10.1016/j.procs.2015.04.172

sensitive PageRank schemes require computing many PageRank vectors, intensifying the need for faster methods for computing PageRank. Previous approaches to accelerating the PageRank computation exploit general sparse graph techniques and use values from the current iteration as they become available, rather than using only values from the previous iteration. They also suggest that exploiting the "bow-tie" structure of the web would be useful in computing Page-Rank. However, the method they suggest is not practical, as ordering the web matrix according to this structure require depth-first search, which is prohibitively costly on the web. More recently, it has been suggested that using successive intermediate iterates to extrapolate successively better estimates of the true PageRank values. However, the speedups from the latter work are modest for the parameter settings typically used for PageRank[5,6,7].

This paper proposes exploiting the detailed typology of the web graph. Analysis of the graph structure of the web has concentrated on determining various properties of the graph, such as degree distributions and connectivity statistics, and on modelling the creation of the web graph[11,2]. However, this research has not directly addressed how this inherent structure can be effectively exploited to speed up link analysis. Raghavan and Garcia-Molina[10] have exploited the hostname (or more generally, url)-induced structure of the web to efficiently represent the web graph. In this paper, we directly exploit this kind of structure to achieve large speedups compared with previous algorithms for computing PageRank by using Shark-Search. Using Shark-Search the crawler search more extensively in areas of the web in which relevant pages have been found. At the same time, the algorithms discontinues search in regions that do not yield relevant pages. Shark search offers two main improvements. It uses a continuous valued function for measuring relevance. In addition, Shark-Search has a more refined notion of potential scores for the links in the crawl frontier.

The rest of paper is organized as follows: Section 2 describes background and related work in area. In section 3 describes the experimental setup. In section 4 proposes a novel shark-search engine. Section 5 provides the conclusion and future work.

2. Background and related works

In this section, we deal with the previous specific algorithms for scheduling the visits to the Web pages. We started with a large sample of the links data collection that was used to build a web graph and run a crawler simulator to ensure identical conditions during the experiments. We describe and evaluate three different crawling algorithms that we have implemented within our evaluation framework: Breadth-First, Best-First, Page-Rank.

2.1. Breadth – First

Under this strategy, the crawler visits the pages in breadth-first ordering. It starts by visiting all the home pages of all the "seed" websites, and web page heaps are kept in such a way that new pages added go at the end. This is the same strategy tested by Najork and Wiener, which in their experiments showed to capture high quality pages first. A Breadth-First crawler is the simplest strategy for crawling. This algorithm was explored as early as 1994 in the web crawler. It uses the frontier as a FIFO queue, crawling links in the order in which they are encountered. Note that when the frontier is full, the crawler can add only one link from a crawled page. Breadth-First is used here as a baseline crawler; since it does not use any knowledge about the topic, we expect its performance to provide a lower bound for any of the more sophisticated algorithm [8, 9].

```
BFS (topic, starting_urls){
   for each link (starting_urls){
      Enqueue (frontier, link, 1);
}
While (visited<MAX_Pages){
          link:= dequeue_top_link(frontier);
          doc:= fetch(link);
          Score:= sim(topic, doc);
          Enqueue (frontier, extract_links(doc), score);
```

```
            If(#frontier > MAX_BUFFER){
                Dequeue_bottom_links (frontier);
            }
        }
}
```

2.2. Best-First

The basic idea in Best-First crawler[6,15] is that given a frontier of links, the best link according to some estimation criterion is selected for crawling [10, 11]. BFSN is a generalization in that at each iteration a batch of top N links to crawl are selected. After completing the crawl of N pages, the crawler decides on the next batch of N and so on. Typically an initial representation of the topic, in our case a set of keywords, is used to guide the crawl. More especially this is done in the link selection process by computing the lexical similarity between a topic's keywords and the source page for the link. Thus the similarity between a page p and the topic is used to estimate the relevance of the pages pointed by p. the N URLs with the best estimates are then selected for crawling. Cosine similarity is used by the crawlers and the links with the minimum similarity score are removed from the frontier if necessary in order not to exceed the buffer size MAX_BUFFER. The sim() function returns the cosine similarity between topic and page:

$$sim(q;p) = P\ k^2\ q\backslash p\ wkqwkp\ qP\ k^2pw^2kpPk^2qw^2kq \quad (1)$$

Where q is the topic, p is the fetched page, and wkp (wkq) is the frequency of term k in p(q).

2.3. Page- Rank

PageRank, relies on the uniquely democratic nature of the web by using its vast link structure as an indicator of an individual page's value. PageRank is a **probability distribution** used to represent the likelihood that a person randomly clicking on links will arrive at any particular page. PageRank can be calculated for collections of documents of any size. It is assumed in several research papers that the distribution is evenly divided between all documents in the collection at the beginning of computational process. The PageRank computations requires several passes, called "iterations" through the collection to adjust approximate PageRank values to more closely reflect the theoretical true value. A probability is expressed as a numeric value between 0 and 1. A 0.5 probability is commonly expressed as a "50% chance" of something happening. Hence, a PageRank of 0.5 means there is a 50% chance that a person clicking on a random link will be directed to the document with the 0.5 PageRank.

```
PageRank (topic, starting_urls, frequency){
        Foreach link (starting_urls) {
            Enqueue(frontier, link);
}
While (visited< MAX_PAGES) {
        If(multiplies(visited, frequency)) {
        recomputed_scores_PR;
    }
link :=deueue_top_link(frontier);
doc:=fetch(link);
score_sim:=sim(topic, doc);
Enqueue(buffered_pages, doc, score_sim);
If (#buffered_pages>=MAX_BUFFER) {
        Dequeue_bottom_links(buffered_pages);
    }
Merge(frontier > extract_links(doc), score_PR);
if(#frontier > MAX_BUFFER) {
Dequeue_bottom_links(frontier);
    }
```

```
    }
}
```

3. EXPERIMENTAL SETUP

In the following sections, we investigate the structure of the web, introducing an algorithm for computing PageRank, and discuss the speedup this algorithm achieves on realistic datasets. Our experimental setup is as follows. We used two datasets of different sizes for our experiments. The STANDFORD/BERKELEY link graph was generated from a crawl of the standford.edu and berkely.edu domains created in December 2002 by the StandfordWebBased Project. This link graph (after dangling node removal, discussed below) contains roughly 683500 nodes, with 7.6 million links, and requires 25MB of storage. We use STANFORD/BERKELEY while developing the algorithms, to get a sense for their performance. For real-world performance measurements, we use the LARGEWEB link graph, generated from a crawl of the Web that had been created by the StanfordWebBase project in January 2001. The full version of this graph, termed FULL-LARGEWEB, contains roughly 290M nodes, just over a billion edges, and requires 6GB of storage. Many of these nodes are dangling nodes (pages with no outlink), either because the pages genuinely have no out links, or because they are pages that have been discovered but not crawled. In computing Page-Rank, these nodes are excluded from the web graph until the final few iterations, so we also consider the version of LARGEWEB with dangling node removed, termed DNLARGEWEB, which contains roughly 70M nodes, with over 600M edges, and requires 3.6GB of storage. The link graphs are stored using an adjacency list representation, with pages represented as 4-byte integer identifier. On an AMD Athon 1533MHz machine with a 6-way RAID-5 disk volume and 3.5GB of main memory, each iteration of PageRank on the 70M page DNR-LARGEWEB dataset takes roughly 7 minutes. Given that computing PageRank generally requires up to 100 iterations, the need for fast computation methods for larger graphs with billion of nodes is clear. Our criteria for determining the convergence of the algorithms that follow uses the norm of the residual vector; i.e., $\|Ax - x\|$. We refer the reader to[11] for a discussion of why the L_1 residual is an appropriate measure for measuring convergence. In this paper, we study distributed real-time database system which consists of a group of main memory real-time database connected by high speed network. Main memory database have been increasingly used for real-time data management because of the high performance of memory accesses and the decreasing main memory cost. The demands of management of the huge distributed and shared data resource become more and more important. The Data Grid is a solution for the above problem.

A data grid composed of a large number of distributed computation and storage resource to facilitate the management of the huge distributed and sharing data resource efficiently. It is a serious challenge to ensure efficient access to such huge and widely distributed data in a data grid. Transaction arrives according to a Poisson process with rate Arrival Rate, and each transaction has an associated firm deadline, assigned as described below. Each transaction randomly choose a site in the system to be the site where the transaction originates and then forks off cohorts at all the sites where it has to access data. Transaction in a distributed system can execute in either sequential or parallel fashion. This is determined by parameter TransType. The distinction is that cohorts in a sequential transaction execute one after another, whereas cohorts in parallel transaction are started together and execute independently until commit processing initiated. We consider both sequential and parallel transaction in the study. Note, however, that replica updaters belonging to the same cohort always execute in parallel. Upon arrival, each transaction T is assigned a firm completion deadline using the formula

$$\mathbf{Deadline_T = Arrival\ Time_T + Slack\ Factor \times\ R_T}$$

Where $Deadline_T$; $Arrival\ Time_T$; and R_T are the deadline, arrival time, and resource time respectively of transaction T ; while Slack Factor is a slack factor that provides control of the tightness/ slackness of transaction deadlines. The resource time is the total service time at the resource (including CPUs and disks) at all sites that the transactions require for its execution in the absence of data replication. It is important to note that while transaction resource requirement are used in assigning transaction deadlines, the system itself lacks any knowledge of these requirements in our model since for many application it is unrealistic to expect such knowledge.

4. Shark-Search using PageRank

We now present the BlockRank algorithm that exploits the empirical findings of the previous section to speed up the computation of PageRank. This work is motivated by and builds on aggregation/disaggregation techniques[5] and domain decomposition techniques[6] in numerical linear algebra. Step 2 and 3 of the BlockRank algorithm are similar to the Rayleigh-Ritz refinement techniques. We begin with a review of PageRank in section 4.1.

4.1. Preliminaries

In this section, we summarize the definition of PageRank and review some of the mathematical tools we will use in analyzing and improving the standard iterative algorithm for computing page rank. Underlying the definition of PageRank is the following basic assumption. A link from a page u ∈ web to a page v ∈ web can be viewed as evidence that v is an important page. In particular, the amount of importance conferred on v by u is proportional to the importance of u and inversely proportional to the number of pages u points to. Since the importance of is itself not known, determining the importance for every page I ∈ web requires an iterative fixed-point computation.

The PageRank of a page i is defined as the probability that at some particular time step $k > K$, the surfer is at page i. For sufficiently large K, and with minor modification to the random walk, this probability is unique, illustrated as follows. Consider the Markov chain induced by random walk on G, where the states are given by the nodes in G, and the stochastic transition matrix describe the transition from i to j is given by P with $P_{ij} = 1/deg\,(i)$.

4.2. Shark_PageRank Algorithm

We propose several improvements to the original fish-search algorithm in order to overcome this limitation. The results in a new algorithm, called the "shark_page search" algorithm, that while using the same simple metaphor, leads to the discovery of more relevant information in the same exploration time. One immediate improvement is to use, instead of binary (relevant/irrelevant) evaluation of document relevance, what we will call hereafter a *similarly engine* in order to evaluate the relevance of document to a given query. Such an engine analyze two document dynamically and return a "fuzzy" score, i.e., a score between *0* and *1* (*0* for no similarity whatsoever, *1* for perfect "conceptual" match) rather than a binary value. A straightforward method for building such an engine is to apply the usual vector space model. It has been shown that these kinds of techniques give better result than simple string or regular-expression matching. The similarity algorithm can be seen as orthogonal to the fish-search algorithm. We will assume in the rest of the paper that such an engine is available and that for any pair query, document, (**q,d**), it return a similarity score **sim(q,d)** between *0* and *1*.

This first improvement has a direct impact on the priority list. We use "fuzzy" relevance score, giving the child an *inherited score,* thus preferring the children of a node that has a better score. This information can be propagated down the descendents chain, thus boosting the importance of grandchildren for a relevant node over the grandchildren of an irrelevant node. The children of irrelevant node use their parent's inherited score for calculating their own inherited score by multiplying it by some decay factor **d**, whose role is comparable to Marchiori's "fading" factor. Thus, suppose document **X** and **Y** were analyzed and that **X** has higher score. Suppose further that the children of both **X** and **Y** have a null score, and the algorithm now has to select the most promising of their grandchildren. Since both their score are multiplied by d^2, the shark search chooses **X's** grandchildren first(which seems intuitively correct). Stepping further away from **X**, into the great-grandchildren zone, would bring **Y's** descendent back to consideration, given an appropriate selection of **d**. This continuous value behavior of score inheritance is much more natural than the Boolean approach of the fish-search.

```
Shark_PageRank(topic, starting_urls) {
for each link (stating_urls){
set_depth (link, d)
Enqueue (frontier, link);
}
```

```
while(visited < MAX_PAGES){
link := Dequeue_top_link(frontier);
doc := fetch(link);
doc_score := sim(topic, doc);
if(depth(link) > 0){
  for each outlink (extract_link(doc)){
    score = (1-r) * neighbourhood_score(outlink) + r *inherited_score(outlink);
if(doc_score > 0){
set_depth(outlink, d);
} else {
set_depth(outlink, depth(link) – 1);
}
enqueue(frontier, outlink, score);
}
If(#frontier > MAX_BUFFER){
 deueue _bottom_links(frontier);
          }
        }
    }
}
```

A more significant improvement consist of refining the calculation of the potential score of the children not only by propagating ancestral relevance score deeper down the hierarchy, but also by making use of the meta-information contained in the links to documents. We propose to use the hints that appear in the parent document, regarding a child. The anchor text of the link is the author's way to hint as to the subject of the linked document. A surfer on the web encountering a page with a large amount of links will use the anchor text of the links in order to decide how to proceed. Some automated tools also take advantage of that information. This approach however can fail in poorly styled HTML document, in which anchor texts consist only of "click here" or "jump here", or when the anchor information is a picture without ALT information. To remedy this problem we suggest using the close textual context of the links, and combining the information extracted from it with the information extracted from the anchor text. To reduce the risk of mistakenly giving a high potential score to a node due to a textual context that actually belongs to another node linked in the same paragraph (context boundaries are difficult to identify), we suggest a small fix. If a node has a relevant anchor text, the score of a textual context is set to the maximum (value of 1), thus giving it an edge over neighbouring links. We claim that a policy that selects the children with the most promising anchor and anchor context information is preferable to arbitrarily selecting the *first* set of children.
These heuristics are so affective in increasing the differentiation of the score of the documents on the priority list, that they make use of the width parameter redundant – there is no need to arbitrarily prune the trees. Therefore, no mention is made of the *width* parameter in the shark-search algorithm that is more formally described in that only the delta from the fish-search algorithm is given.

5. Conclusion and Future Work

In this paper, we have proposed an improved version of one of the first dynamic search algorithm for the Web, the "fish search" algorithm. Our more aggressive Shark-Search algorithm overcomes some limitation of the fish-search by analyzing the relevance of documents more precisely and, more importantly making a finer estimate of the relevance of neighbouring pages before they are actually fetched and analyzed. These improvements enable to save precious communication time, by fetching first document that are most probably relevant or leading to relevant document, and wasting time on garden paths. We implemented both the fish and Shark-search algorithms in a dynamic tailorable site mapping tool, and experimentally verified that significant improvement were achieved in terms of the total of relevant information discovered in the same small amount of time. We believe that the shark-search algorithm can thus advantageously replace its ancestor in applications that require dynamic and precise searches in limited time spans.

References

1. Soumen chakrabarti, Martin van den Berg, and Byron Dom. Focused crawling: a new approach to topic-specific web resource discovery. *Computer Networks*, 31(11-16): 1623-1640, 1999.
2. Adam kilgarriff and Gregory Grefenstette. Introduction to the special issue on the web as corpus.
3. Internet archive project. http://www.archive.org/, 2004**.**
4. Ricardo Baeza-Yates and Carlos Castillo. Balancing volume, quality and freshness in web crawling. In soft computing systems- Design, management and applications, pages 565-572.
5. Google search engine. http://www.google.com/, 2004.
6. Steve Lawrence and C. Lee Giles. Searching the World Wide Web. Science, 280(5360): 98-100, 1998.
7. Jacob Nielsen. Statistics for traffic referred by search engine and navigation directories to useit. http://www.useit.com/about/searchreferrals.html, 2003.
8. Lawrence Page, Sergey Brin, Rajeev Motvani, and Terry Winograd. The Pagerank citation algorithm: bringing order to the web. In *Proceeding of the seventh conference on world wide web*, Brisbane, Australia, April 1998.
9. Gerard Selton*. The SMART retrieval system experiments in automatic document processing*. Prentice-Hall, 1971.
10. Arvind Arasu, Junghoo Cho, Hector Garcia-Molina, Andreas paepcke, and Sriram Raghavan. Searching the web. *ACM Transaction on internet technology (TOIT),* 1(1):2-43, August 2001.
11. Eytan Adar and Bernardo A. Huberman. The economics of web surfing. In *Poster Proceedings of the Ninth Conference on World Wide Web,* Amsterdam, Netherland, May 2000.