2007 IEEE/WIC/ACM International Conference on Web Intelligence

FICA: A Fast Intelligent Crawling Algorithm

Ali Mohammad Zareh Bidoki
University of Tehran
Zare_b@ece.ut.ac.ir

Nasser Yazdani
University of Tehran
Yazdani@ut.ac.ir

Pedram Ghodsnia
University of Tehran
Pedram@pedram.ir

Abstract

*Due to the proliferation and highly dynamic nature of the web, an efficient crawling and ranking algorithm for retrieving the most important pages has remained as a challenging issue. Several algorithms like PageRank [13] and OPIC [1] have been proposed. Unfortunately, they have high time complexity. In this paper, an intelligent crawling algorithm based on reinforcement learning, called FICA is proposed that models a real surfing user. The priority for crawling pages is based on a concept which we name as logarithmic distance. FICA is easy to implement and its time complexity is O(E*logV) where V and E are the number of nodes and edges in the web graph respectively. Comparison of the FICA with other proposed algorithms shows that FICA outperforms them in discovering highly important pages. Furthermore, FICA computes the importance (ranking) of each page during the crawling process. Thus, we can also use FICA as a ranking method for computation of page importance. We have used UK's web graph for our experiments.*

1. Introduction

One of the most challenging issues for web search engines is finding high quality web pages or pages with high popularity for users. To make the web more interesting and productive, we need an efficient ranking algorithm for crawling and searching. In [6], it has been shown that search engines do not index the entire Web. Therefore, we have to focus on the most valuable and appealing pages. To do this, a better crawling technique is required and a more efficient mechanism has to be applied. This enables search engines to present the most relevant pages to the user in response to her query.

Crawling algorithms use a ranking mechanism to discover important pages. In other words, a ranking algorithm is applied to the current web graph and pages with higher ranks will have a higher priority for crawling.

Ranking methods are usually based on links between web pages or the graph structure of pages. Their main strength comes from using content of other pages to rank current pages [8]. Examples of connectivity based ranking algorithms are PageRank [13], HITS [10] and OPIC [1].

Unfortunately, the current ranking algorithms usually have high complexity and low throughput. Obviously, these drawbacks are inherited in the crawling processes which use them.

We propose a crawling algorithm, called FICA (Fast Intelligent Crawling Algorithm), based on reinforcement learning [14] in which "punishment" is calculated using the distance between pages. We define this distance as the number of "average clicks" between two pages as defined in [11]. In our method, the aim is to minimize sum of received punishments (distance) by the crawler agent so that a page with a low distance to have the highest priority for crawling.

On currently crawled page p with distance d_p, the distance of each of its child nodes, q, is computed as $d_q = \alpha * \log(O(p)) + (1-\alpha) * d_p$ where O(p) shows out-degree of p and α is the learning rate of the crawler agent.

A nice property of our method is that it tries to model a user surfing the web randomly. Initially, a user browsing the web does not have any background about the pages and clicks based on the current status of each page. As time goes on, the user selects a page (clicks on a link) based on both her background and the current content of each page. As she continues, she accumulates more knowledge from the environment and other web pages, and improves her selection.

A major contribution of this paper is an efficient crawling algorithm that finds hot pages faster (earlier) than former algorithms. The method also computes the importance of web pages during the crawling process with low complexity and less resource while crawling each page only once. In other words, after the crawling process ranking of the pages will be available too. This means that ranking is performed online or while

0-7695-3026-5/07 $25.00 © 2007 IEEE
DOI 10.1109/WI.2007.91

crawling. This is why we talk about ranking and crawling simultaneously in this paper.

UK's web graph (.uk websites) has been used to evaluate our algorithm. The complexity of our solution is at most O(E*logV) where V and E are the number of nodes (vertices) and edges in the web graph respectively. We have compared our algorithm with other crawling algorithms. Our algorithm outperforms other algorithms in throughput, i.e. it finds important pages faster than others. Furthermore, we used FICA as a ranking algorithm and compared it with PageRank and OPIC. Interestingly, its ranking similarity against PageRank is 0.61.

The next section reviews background and related work. Section 3 discusses our solution, FICA. Experimental analysis and comparison to some of the well-known algorithms come in section 4. Section 5 describes using FICA as a ranking algorithm and finally, our conclusion and future areas of research are presented in section 6.

2. Background and related work

There are two broad categories of crawling algorithms. Some algorithms are based on ranking algorithms like PageRank [13] and some are based on scheduling crawling algorithms like OPIC [1] and the Breadth-first algorithm [12]. In PageRank based crawling the pages with higher ranking will have higher priority for retrieval from the web.

PageRank has been designed such that the known relationships between web pages is taken into account. For example, if page p_1 has a link to page p_2, then, p_2's subject is probably interesting for p_1's creator. Therefore, the number of input links to a web page shows the interest degree of the page to others. Clearly, the interest degree of a page increases with the growing number of input links. Furthermore, when a web page receives links from an important page, then, naturally, it should have a high rank. Therefore, PageRank of a web page corresponds to the weighted sum of input links.

Let pages on the Web to be denoted as *1,2,…,n, and* O(*i*) denotes the number of outwardding (outgoing) links from page i and *B*(*i*) denotes the set of pages that point to page *i*. The PageRank of page j, denoted by *r*(j), is given by:

$$r(j) = (1-d)/n + d * \sum_{i \in B(j)} r(i)/O(i) \qquad (1)$$

Thus, the PageRank of page j is equal to sum of the input page ranks divided by their out-degree. Dividing input pages ranks by their outputs degree, O(i), has two effects. First, it distributes a PageRank to all outputs fairly and, secondly, it normalizes sum of each page effects and rank vector to one. Parameter d, damping factor, is used to guarantee the convergence of PageRank and to remove effects of sink pages, pages with no outputs. Thus, in this way, each web page will have outgoing links to every single web page.

PageRank can be written as a linear relation $r = A^T * r$ where *r* is an n dimensional vector [*r*(1), *r*(2),..., *r*(n)] and elements a_{ij} of matrix *A* are given by $a_{ij} = 1/O(i)$ if page *i* points to page *j* and $a_{ij}=0$ otherwise. Our goal is to compute r that is the eigen vector of A^T for eigen value one [2].

This mechanism is equivalent to the *Random Surfer* behaviour, a person who surfs the Web by randomly clicking links on the visited pages. When the user reaches a page with no output links s/he will jump to a random page. Therefore, when a user is in a web page, with probability of d s/he will select one output link randomly or will jump to other web pages with the probability of 1-d.

In [12], 328 million real web pages using breadth-first ordering has been crawled. Experimentally, it is found that in the breadth-first algorithm the pages with high PageRank will be crawled earlier. The reason is that the most important pages have many input links and we will find them early.

In [5], a comparison between some crawling algorithms including PageRank, Back-Link and breadth-first has been done. It was found that crawling based on PageRank finds the most important web pages faster than others.

Abiteboul et al. [1] proposed an algorithm called OPIC, On-line Page Importance Computation. In their method, each page has a cash value that is distributed equally to all output links (initially all of pages have the same cash equal to 1/n). This is similar to PageRank while it is done in one step. In every state, the crawler will download web pages with higher cashes and when a page is downloaded its cash will be distributed among the pages it points to. Experiments were done on a synthetic web graph including at most 600,000 nodes with the power law distribution [15]. There is no comparison between OPIC and other crawling strategies. Unfortunately, in this method, each page will be downloaded many times that will increase crawling time.

In [4], a site based method named "largest sites first" has been proposed. In this method, sites with the larger number of pending pages have higher priority for crawling. It is found that this algorithm is better than the breadth-first.

3. FICA

The breadth-first algorithm traverses the graph by following its links. The distance of each currently crawled page from the root (starting points) is always less than or equal to that of the uncrawled pages. Consider the crawling tree produced in the breadth-first manner. The breadth-first algorithm traverses the tree level by level. For instance, in Figure 1, pages will be crawled in the order of p, q, r, s, t… w. Our algorithm is based on the breadth-first algorithm with a new definition of distance between web pages [11]. We give the following definitions to better clarify our idea:

Definition 1. Link weight: If page i points to page j then the weight of the link between i and j is equal to $Log_{10}O(i)$ where O(i) denotes i's out-degree (number of outward links).

Definition 2. Logarithmic distance: The distance between pages i and j is the weight of the shortest path (the path with the minimum value) or sum of link weights in the shortest path from i to j. We call this **logarithmic distance** and denote it with d_{ij}. We denote the logarithmic distance between the root and page i with d_i.

For example, in Figure 1, the weight of outward links in pages p, q and s are equal to log2, log3 and log4 respectively. The distance between p and t is log2+log3 and between p and v is log2+log2. Thus, whereas both t and v are the same number of links away from p (two clicks), v is closer to p in terms of logarithmic distance ($d_{pv}<d_{pt}$).

For the rest of this paper we use the terms "distance" and "logarithmic distance" interchangeably unless stated otherwise.

If a crawled page i has distance d_i from the root page, by using definition 2 the distance (or logarithmic distance from the root) of each of its child nodes is computed as equation 2.

$$\mathbf{d_j=d_i+log(O(i))} \quad (2)$$

Where j is a child of i and O(i) denotes i's out-degree (number of outward links).

In our algorithm, called FICA (Fast Intelligent crawling Algorithm), pages considered for crawling are those with lower logarithmic distances from the root. In other words, the priority of pages for crawling is dependent on their logarithmic distance from the root. So the crawling order for Figure 1 will be: p, q, r, v, w, s, t, and u.

Like other algorithms, our method is dependent on the out-degree of nodes in the web graph. FICA follows the web graph like the random surfer model [3] used in PageRank in that each outward link of page i is selected with probability 1/O(i). To show this fact, we define the **rank effect** of i on page j as the inverse product of the out-degrees of pages in the path between i and j. For example, if there is a path with length 3 from i to j like i->k->l->j, then i's rank effect on j is (1/O(i))*(1/O(k))*(1/O(l)). In other words, the probability that a random surfer starting from page i will reach page j through this path is (1/O(i))*(1/O(k))*(1/O(l)). Lemma 1 shows the relationship between rank effect and logarithmic distance.

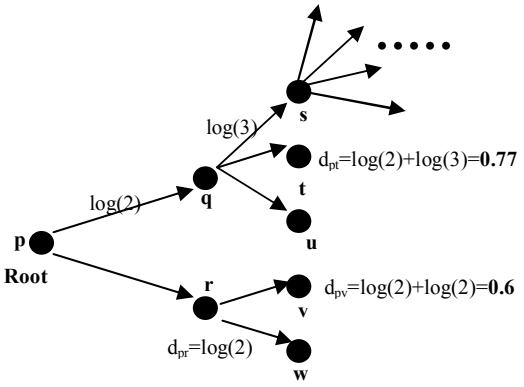

Figure 1: Logarithmic distances in the crawling tree.

Lemma 1: Suppose d_{ij} shows the logarithmic distance between pages i and j and r_{ij} shows i's rank effect on j in the shortest path between i and j, then if, $d_{ij}<d_{ik}$ then $r_{ij} > r_{ik}$.

Proof: We have

$$d_{ij} = \sum_{s\in path(i,j)} \log O(S) = \sum_{s\in path(i,j)} -\log(\frac{1}{O(s)}) =$$

$$-\log \prod_{s\in path(i,j)} \frac{1}{O(s)} = -\log r_{ij}$$

Also $d_{ik}= -\log r_{ik}$ thus when $d_{ij} <d_{ik}$, we will have $r_{ij} > r_{ik}$ □.

Therefore, if the logarithmic distance between i and j is less than that of between i and k then i's rank effect on j is more than it's rank effect on k. In other words, it is more probable that a random surfer starting from i will traverse the shortest path to j than the shortest path to k.

Using equation 2, we propose the following formula which is based on the reinforcement learning algorithm [14] to compute the distance of page j (i links to j).

$$\mathbf{d_j=\alpha *log(O(i)) + (1-\alpha)*d_i} \ , \ 0<\alpha\leq 1 \quad (3)$$

Where α is the learning rate and log(O(i)) is the instantaneous punishment the crawler receives in transition from i to j. In the environment of the web, our aim is to decrease the sum of punishments received by the crawler. We model the learning rate, α as in equation 4 in which t shows time or the iteration

number and β is a static value to control regularity (see section 3). Experiments (discussed in the next section) reveal that if the learning rate α is adjusted properly (it must be slowly decreased) we achieve optimum results and high throughput very fast. When the algorithm starts, we don't have the distance of any page so we initially set α to one and, then, decrease it asymptotically to zero.

$$\alpha = e^{-\beta * t} \quad (4)$$

From the learning view point, the crawler is an agent surfing the web randomly, receiving some punishments from the environment in each step of the way. Its goal is to minimize the sum of punishments. In each state, the agent has to choose the next page to crawl and the page with the minimum received punishment (distance) will be selected. Obviously, we can write equation 3 as the following:

$d_j = \alpha$ *(instantaneous punishment of selection j)+

(1- α)* (punishment received up to page i)

So d_j is the total punishment that the agent receives from selecting page j.

A major contribution of our method is modelling the real user surfing the web. Intuitively, when a user starts browsing from a random page, she does not have any background about the web. As she continues surfing and visiting web pages, she clicks on links based on both her pervious experience and the current status (content) of the web pages she visits. Continuously, she accumulates knowledge that helps her reach her goal and find suitable pages faster. Our algorithm acts in a similar fashion. Initially, the agent has little knowledge about the web pages (environment), hence α =1, as it visits more pages, it slowly learns from the environment (α decreases) and selects further pages better than before. Obviously, its learning rate is high at first and decreases to zero with time.

Equation 3 is used when we reach page *j* for the first time. The question is what happens when we encounter the link of an uncrawled page more than once? In this state, we choose the path from the parent which produces the least distance. For example, suppose page j currently has distance 2.3 and has been reached directly through page i. The crawler then finds page j for the second time through page k.Suppose page k has distance 0.4 and an out-degree of 10. Then the distance of page j through page k is calculated as 1.4. Since this number is smaller than the previously found 2.3, we choose it as the new distance value for page j. Thus, the distance formula changes to equation 5 where B(j) is the set of pages that point to j (Figure 2).

$$\mathbf{d_j} = \min_i (\alpha * \mathbf{log(O(i))} + (1-\alpha) * \mathbf{d_i}), i \in B(j) \quad (5)$$

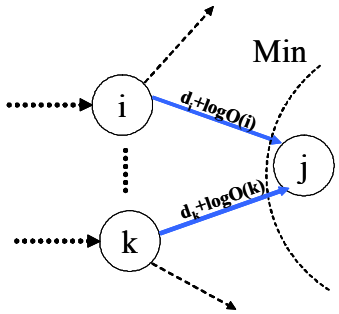

Figure 2: Logarithmic distance for page j.

The pseudo code of the proposed crawling algorithm is shown below. This algorithm is similar to Dijkstra's algorithm for computing single-source shortest paths.

Algorithm 1. FICA Crawling algorithm

```
//D is the distance vector
   D ← {∞, ∞, ∞, …}
Input: starting_url , K=250,000
t=0, β = 0.1, Size=0
distance=0;
enqueue(URL_queue, (starting_url,distance))
While (not empty(URL_queue))
     (url,distance) = dequeue(URL_queue)
     Size=Size+1
     t=Size div K
     α =exp(- β * t)
     distance = α *log(O(url))+(1- α )* distance
     crawl_page(url)
     for each child u of url
       if(distance <D[u])
        if ( u ∉ crawled_pages)
          enqueue(URL_queue, (u, distance))
        D[u]= distance
End while
```

The URL-queue is a priority queue, containing pages waiting to be crawled in which the priority is the distance value. In this mechanism, the distance of each page, in addition to its URL, is inserted into the URL-Queue and the pages are sorted by their distances in ascending order. The distance value of page j is equal to the logarithmic distance between the root and page j. In this algorithm, D is the distance vector in which D[j] shows the current distance value of node j and is set to a big value initially. The enqueue(queue, (element,d)) function inserts an element with the distance d into it's position in the sorted queue (elements are sorted according to their distances). Dequeue(queue) removes the element at the beginning of the sorted queue and its distance and returns them.

The number of iterations or t is incremented after crawling every K (250,000) pages. β value must be

set properly for learning rate justification. We found that for K=250,000, β=0.1 is suitable.

By finding a new uncrawled page, we insert it into the URL_queue with its received distance if it is not in the URL_queue already. Otherwise, we change its distance to the new one if the new distance is less than the current distance. In this way, every page is crawled once and only its distance is changed. This accelerates the running time and the algorithm finds new important contents faster.

We use a binary tree for queue management, then, by lemma 2, the complexity of our algorithm in the average case is reduced to O(E*logV). While the complexity of PageRank in the worst case is O(V*E) in which V and E are the number of nodes and edges respectively. However based on [7] it is around O(100*E) i.e. 100 iterations for an acceptable ranking is sufficient. Since we run PageRank for V/K times (algorithm 2), the complexity of the crawling based on PageRank is around O(100*E*V/K). Thus, FICA is faster than crawling methods based on PageRank by a factor of (100*V)/(K*LogV). For example, if we have 10 billion pages ($V = 10^{10}$) and we run the PageRank algorithm for 200 times ($V / K = 200$), the complexity of any crawling method based on PageRank is O(20000*E) while FICA's complexity is O(10*E). Furthermore, in pure crawling algorithms like OPIC each page is crawled many times while in FICA each page is crawled once and only its distance is changed.

Our solution is fast and requires small memory for computation, since we do not need to store the matrix of the web graph. The algorithm also accomplishes crawling and ranking simultaneously (as opposed to other algorithms which carry out the crawling and ranking processes sequentially). This means that after running the FICA algorithm we will also have relatively acceptable ranks of all pages at the same time (see section 4).

Lemma 2: Complexity of the FICA algorithm in the average case is *O(E*logV)*.

Proof: The average time of insertion or deletion for a binary tree with size M is logM. Each edge of the graph is traversed at most once and for each such edge the priority queue may be updated. Thus in the average case, we have E insertions or deletions so that the time of all insertions or deletions is O(E*logV).□

4. Experimental results

For the evaluation of FICA, we used UK's web [16] which includes over 18 millions web pages all those ending with .uk. Our goal was to compare FICA with other crawling algorithms and evaluate which one of them finds more important pages faster. We compared our algorithm with the following crawling algorithms:

Breadth-first: The crawling process is done in the breadth-first order. Initially, the algorithm starts with some starting URLs as the roots of the crawling tree.

Back-link count: In this algorithm, pages with more input links are crawled first [5], that is, pages with more input links have higher ranks.

Partial PageRank: This method uses the PageRank algorithm [13] on the web pages seen so far and crawls the pages with higher PageRanks first.

OPIC: In this algorithm, all pages start with the same amount of "cash" [1]. Every time a page is crawled, its cash is distributed to its outward links. In each step the next page for crawling is the one with the highest amount of cash up to now (the priority is cash). Initially, we start crawling the web with some starting URLs. In our experiment we randomly selected 1000 pages as starting URLs (roots). Because α is less than 1, dependency of our algorithm on the root selection is very low. For example, if page j is 10 links farther away than the root page (10 clicks), then distance effect of the root on this page is a factor of α^{10}.

Every time by crawling K new web pages (K is set to 250,000), we run one of the above ranking algorithms (each algorithm will run 72 times[1]). After that, we sort the web pages in the queue according to the produced ranking. This process continues until a specified portion of the web is crawled. The general crawling and ranking algorithm is as algorithm 2 below. All methods run this with their own ranking criteria [5]. Unlike other ranking algorithms, FICA and OPIC are scheduling algorithms and differ in that they do not require an additional ranking stage.

Algorithm 2. The general crawling algorithm

```
Input: starting_url, K=250,000
enqueue(URL_queue, starting_url)
 while (not empty(URL_queue))
    url = dequeue(URL_queue)
    crawl_page(url)
    for each child u of url
     if(u∉ URL_queue and (u)∉ crawled_pages)
               enqueue(URL_queue, u)
     if(crawled_pages.count()%K==0)
               reorder_queue(URL_queue)
 End while
```

The enqueue(queue, element) function appends an element to the end of the queue, dequeue(queue) removes the element at the beginning of the queue and returns it and reorder_queue(queue) reorders the queue using one of the ranking algorithms below:

1. Breadth first

do nothing

[1] 18millions/250,000=72.

2. Back-link count

For each u in URL_queue

Back-link count[u] = number of u input links

Sort URL_queue by backlink count[u]

3. PageRank

Solve the following set of equations:

$$PR[u] = (1 - 085)/n + 0.85 * \sum_{p_i \in B[u]} \frac{PR[p_i]}{O[p_i]}$$

Where B[u] shows list of pages linking to u and O[p_i] is the number of links in the page p_i

Sort URL_queue by $PR(u)$

OPIC algorithm runs in a different procedure [1]. To be fair we set OPIC to crawl each page only once but the received cash is changed every time we visit its link in other pages. The aim of crawling is to find hot pages. To do this purpose we choose the PageRank algorithm as a benchmark. First, ranks of all web pages are computed using the PageRank algorithm on the entire graph. In a set of K pages, gathered from running a crawling algorithm, a page is hot if it exists in the first K hot pages of the benchmark ranking. Clearly, the algorithm that retrieves more hot pages will be better than others. We define **throughput** at each step as the fraction of crawled hot pages to all hot pages that can be discovered.

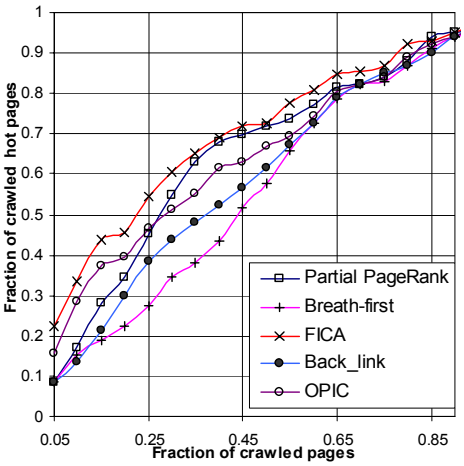

Figure 3: Throughput of crawling algorithms where the benchmark ranking is PageRank.

We compared the aforementioned algorithms with FICA in Figure 3 for 18 million web pages. The damping factor of PageRank and β were set to 0.85 and 0.1 respectively. As Figure 3 shows, FICA outperforms other algorithms specially in the early stages. For example, when 15% of pages are crawled, FICA finds about 43% of hot pages whereas partial PageRank and OPIC find 28% and 37% respectively. In comparison to PageRank and OPIC, FICA have 8% increase in the throughput with very low complexity while our algorithm uses less memory and does not need to store the matrix of the web graph.

Modelling a real user browsing on the web is a major advantage of FICA. In pursuit of suitable pages, the user clicks on new links and visits new pages, accumulating knowledge about the environment (the web) during the process. Thus, each new click is done with more information through better selection.

In reinforcement learning algorithms, the learning rate (α) plays an important role in convergence of the system. We compared the throughput of FICA for different learning functions, that is, the exponential (equation 4) ,x^{-1}, power law [15] ($x^{-2.1}$) and linear (x) functions where x is $\beta * t$. We found that the exponential function is better than others in terms of throughput. In other words, the best function for learning rate is the one with the fastest decreasing slope. These functions model user behaviour better than others. A user learns a lot from the system early on, and as she proceeds, the rate of her learning slowly decreases. Also we considered the throughput of FICA for different β in the exponential learning rate. And found β**=0.1** is the best selection. This value causes the learning rate to decrease more slowly and better resembles the user behaviour. Because of space limitations we removed these experiments from the paper.

5. FICA as a ranking algorithm

As mentioned earlier, FICA has the advantage of producing a ranking of web pages while crawling. Then after the crawling process, we can simply use values in the distance vector as criteria for ranking. That is, pages with lower distances will have higher ranks. To illustrate the validity of this ranking, we used Kendall's τ metric [9] which shows the correlation between two ranked lists. Two identical lists have τ=1, while two totally uncorrelated lists have τ=0 and reversed lists have τ=-1.

We calculated this metric for ranked lists produced by each crawling algorithm compared to the ranked list produced by PageRank (benchmark ranking). Because of complexity of Kendall computation on full list, we chose 30000 pages in uniform manner randomly from each ranked list.

FICA and OPIC have the highest Kendall factors, i.e. 0.61 and 0.62 respectively. Thus FICA algorithm like OPIC is suitable for ranking of web pages. As we discussed, the major contribution of this paper is an efficient crawling algorithm that finds hot pages faster (earlier) than former algorithms. Furthermore, this method computes the importance of web pages during

the crawling process. In other words, after the crawling process ranking of the pages will be available too.

Table 1: Kendall factor of crawling algorithms against PageRank.

Algorithm	Kendall's Tau
Back Link	0.09
Breadth-first	0.11
Partial PageRank	0.18
FICA	**0.61**
OPIC	0.62

6. Conclusion and future work

We proposed a crawling algorithm based on reinforcement learning called FICA. The priority of pages in the crawling process is the logarithmic distance from the root pages (starting urls). Our algorithm models a real user behavior surfing the web. When a user randomly browses the web, she selects each new page based on her background from previous pages and the current status (content) of the current web page.

A major contribution of the paper is an efficient crawling algorithm that finds hot pages faster (earlier) than the previously proposed algorithms. Meanwhile, page importance is computed during the crawling process. In other words, after crawling process we will have an acceptable ranking of pages too.

FICA is fast and easy to implement with low time complexity, around O(E*logV) in the average case. Furthermore, we do not need to store the matrix of the web graph and only a vector to store the distance of each page is sufficient. We used UK's Web to evaluate the proposed algorithm. In comparison with other algorithms like PageRank and OPIC, FICA have higher throughput, which is defined as the fraction of crawled hot pages to all hot pages in each step. We used FICA as a ranking method for computing the importance of pages. The Kendall factor between rankings produced by FICA and PageRank is 0.61. Currently, via the idea used in FICA, we have been proposed a ranking algorithm called DistanceRank [17] that outperforms other ranking algorithms. Evaluation of FICA for larger graphs and proposing an adaptive version of FICA that adjusts dynamically with changes in the web graph remain as future work.

7. References

[1] Serge Abiteboul, Mihai Preda, and Gregory Cobena. Adaptive on-line page importance computation. In Proceedings of the twelfth international conference on World Wide Web, pages 280–290. ACM Press, 2003.

[2] Arvind Arasu, Junghoo Cho, Hector Garcia-Molina, Andreas Paepcke, and Sriram Raghavan. Searching the Web. ACM Transactions on Internet Technology (TOIT), 1(1):2–43, August 2001.

[3] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. In Proceedingsof 7th World Wide Web Conference, 1998.

[4] Carlos Castillo,MauricioMarin, Andrea Rodr´ýguez, and Ricardo Baeza-Yates. Scheduling algorithms for Web crawling. In Latin American Web Conference WebMedia/LA-WEB), pages10–17, Riberao Preto, Brazil, October 2004. IEEE CS Press.

[5] Junghoo Cho, Hector Garc´a-Molina, and Lawrence Page. Efficient crawling through URL ordering. In Proceedings of the seventh conference on World Wide Web, Brisbane, Australia, April 1998.

[6] Antonio Gulli , Alessio Signorini, The indexable web is more than 11.5 billion pages, Special interest tracks and posters of the 14th international conference on World Wide Web, May 10-14, 2005, Chiba, Japan

[7] Taher Haveliwala. Efficient computation of pagerank. Technical Report 1999-31, Database Group, Computer Science Department, Stanford University, February 1999. Available at http://dbpubs.stanford.edu/pub/1999-31.

[8] Monika Henzinger. Hyperlink analysis for the web. IEEE Internet Computing, 5(1):45–50,2001.

[9] Maurice G. Kendall. Rank Correlation Methods. Griffin, London, England, 1970.

[10] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. Journal of the ACM, 46(5):604–632, 1999.

[11] Yutaka Matsuo, Yukio Ohsawa, and Mitsuru Ishizuka, Average-clicks: A New Measure of Distance on the World Wide Web, Journal of Intelligent Information Systems, Vol.20, No.1, pp. 51--62, (2003.1)

[12] Marc Najork and Janet L. Wiener. Breadth-first crawling yields high-quality pages. In Proceedings of the Tenth Conference on WorldWideWeb, pages 114–118, Hong Kong, May 2001. Elsevier Science.

[13] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking:Bringing order to the web. Technical report, Computer Science Department, Stanford University,1998.

[14] R.S. Sutton and A.G. Barto, "Reinforcement Learning: An Introduction",MIT Press, Cambridge, MA, 1998.

[15] Andrei Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet Wiener. Graph structure in the web: Experiments and models. In Proceedings of the Ninth Conference on World Wide Web, pages 309–320.

[16] UK's web graph, http://webgraph-data.dsi.unimi.it/

[17] Ali Mohammad Zareh Bidoki, and Nasser Yazdani. DistanceRank: An intelligent ranking algorithm. Information Processing and Management (2007), doi:10.1016/j.ipm.2007.06.004.