Abstract. The web is a huge and highly dynamic environment which is growing exponentially in content and developing fast in structure. No search engine can cover the whole web, thus it has to focus on the most valuable pages for crawling. So an efficient crawling algorithm for retrieving the most important pages remains a challenging issue. Several algorithms like PageRank and OPIC have been proposed. Unfortunately, they have high time complexity and low throughput. In this paper, an intelligent crawling algorithm based on reinforcement learning, called FICA is proposed that models a random surfing user. The priority for crawling pages is based on a concept we call logarithmic distance. FICA is easy to implement and its time complexity is $O(E \log V)$ where $V$ and $E$ are the number of nodes and edges in the web graph respectively. Comparison of 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. A nice property of FICA is its adaptability to the web in that it adjusts dynamically with changes in the web graph. We have used UK’s web graph to evaluate our approach.

Keywords: Web crawling, web ranking, reinforcement learning, intelligent surfer

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 important and relevant pages to the user in response to her query.

Crawling algorithms usually 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 the sum of received punishments (distance) by the crawler agent so that a page with a low distance will have the highest priority for crawling.
On currently a crawled page \( p \) with distance \( dp \), the distance of each of its child nodes, \( q_i \), is computed as \( d_q = \log(O(p)) + \gamma \cdot d_p \), where \( O(p) \) shows out-degree of \( p \) and \( \gamma \) is the discount factor 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 by, 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, a ranking of the pages will be available too. This means that ranking is performed online or while crawling. This is why we talk about ranking and crawling simultaneously in this paper. Also using the ideas in FICA we have proposed a ranking algorithm called DistanceRank in [17].

UK’s web graph (.uk websites) has been used to evaluate our algorithm. The complexity of our solution is at most \( O(E^* \log V) \) 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. Finally we discuss the adaptability feature of FICA (i.e. that it adjusts dynamically with changes in the web).

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. Section 6 discusses about the complexity issues of FICA. In Section 7, a dynamic version of FICA is explained and finally, our conclusion and future areas of research are presented in Section 8.

2. Background and related work

Crawling has been the subject of extensive research [12,21,24,25] and currently web crawling has been studied in different aspects. Most of them attempt to hold the search engine’s index and repository fresh while covering the most important parts of the web.

There are two broad categories of crawling algorithms. Some 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 are 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 for 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, the PageRank of a web page corresponds to the weighted sum of input links.

Let pages on the web be denoted as \( 1,2,...,n \). \( O(i) \) denotes the number of 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:

\[
 r(j) = (1-d)/n + d \times \sum_{i \in B(j)} r(i)/O(i)
\]

Thus, the PageRank of page \( j \) is equal to the sum of the input pages’ PageRanks divided by their out-degree. Dividing input pages’ PageRanks by their output degree, \( O(i) \), has two effects. First, it distributes a PageRank to all outputs fairly and, secondly, it normalizes the sum of each page’s effects and rank vector to one. Parameter \( d \), the 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

\[
r = A^* \cdot r
\]

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.
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 she will jump to a random page. Therefore, when a user is on a web page, with probability \( d \) she will select one output link randomly or will jump to other web pages with probability of \( 1 - d \).

There is a similar work in [21]. There, a crawling algorithm based on a metric called RankMass has been proposed to find highly important pages after crawling a certain number of pages. The RankMass metric is based on commonly used variations of PageRank such as Personalized PageRank [26] and TrustRank [27] which assume that users’ random jumps are limited to a set of specified pages. These researchers proved the correctness of their algorithm by theoretical analysis and evaluated its performance on 141 millions URLs.

In [12], 328 million real web pages were crawled using breadth-first ordering. Experimentally, it is found that in the breadth-first algorithm the pages with high PageRanks will be crawled earlier. The reason is that the most important pages have many input links and are found 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.

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 method.

Abiteboul et al. [1] proposed an algorithm called OPIC, On-line Page Importance Computation which is similar to ours. In their method, each page has a “cash” value that is distributed equally to all output links (initially all 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 cash 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 leading to increased crawling times.

In [28] a crawling algorithm has been proposed to maintain the freshness of the search engine’s index based on a quality metric using users’ experiences. The pages will be recrawled based on a policy to maximize the quality of the index. Contrary to ours, their goal is not to optimize the crawling process to find high quality pages from a portion of the web.

Dasgupta et al. [22] proposed a crawling algorithm in order to discover newly-arrived content on the web. They measured the overhead of discovering the new content for crawling algorithms. It was shown that by perfect foreknowledge of where to explore for links to new content, it is possible to discover 100% of new content with 9% overhead.

There is a different type of crawler called “focused crawler” that only retrieves pages related to a specific topic. In this type of crawling usually both content and connectivity of the web are used for crawling. For example, pages pointed to by a page of a specific topic are probably related to that topic. These crawlers are used in search engines that cover a particular topic on the web like medicine and agriculture. There are some examples of focused crawlers in [25,29] and also two algorithms based on reinforcement learning in [31,32].

A work has been done in [23] for topical crawlers based on machine learning and evolution. They achieved high performance by combination of explorative and exploitative bias. Also they found that evolutionary crawlers achieve high efficiency and scalability by using concurrent agents.

It is possible to use FICA as a method of focused crawling to retrieve the most interesting pages in a certain topic, but that remains as future work.

There are other work such as [19] and [20] which proposed new models of ranking (not crawling) based on PageRank. Their works are similar to ours in the proposed model on the web graph which is weighted.

In [19] a model called “intelligent surfer” has been proposed for ranking. In this model instead of using a simple random surfer model used in PageRank, a probabilistic combination of link and content has been proposed. In their model, the weights of output links of nodes are not equal and are dependent on the user query. They have found that their algorithm outperforms the PageRank algorithm.

In [20] a new punishment/reward based approach that adds a new dimension to the PageRank model for reducing the effect of the rich-get-richer problem [30] using implicit feedback (click-through data) of visitors, has been proposed. In this approach, in addition to considering the structure of links from a page-
creator’s point of view, the page-visitor’s viewpoint has been utilized as an important parameter to improve the accuracy of the PageRank algorithm.

This paper is the extended version of the work published in [34]. Contrary to previous version, this paper includes global definition of distance according to Reinforcement Learning in addition to the adaptive version of algorithm.

3. FICA

The breadth-first crawling 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 Fig. 1, pages will be crawled in the order of \( p, q, r, s, t, u, v, w \). 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 Fig. 1, the weight of outward links in pages \( p, q \) and \( s \) are equal to \( \log_2 \), \( \log_3 \) and \( \log_4 \) respectively. The distance between \( p \) and \( t \) is \( \log_2 + \log_3 \) and between \( p \) and \( v \) is \( \log_2 + \log_2 \). Thus, whereas both \( t \) and \( v \) have the same number of links away from \( p \) (two clicks), \( v \) is closer to \( p \) in terms of logarithmic distance (\( d_{pt} < d_{pv} \)).

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 Eq. (2).

\[
d_i = \log(O(i)) + d_j
\]

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

In our algorithm (which we call FICA for 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 Fig. 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 \( j \) (\( r_{ij} \)) as the inverse product of the out-degrees of pages in the shortest path between \( i \) and \( j \) (\( path(i,j) \)). For example, if there is a path with length 3 from \( i \) to \( j \) like \( i \rightarrow k \rightarrow l \rightarrow j \), 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.

**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

\[
d_{ij} = \sum_{x \in path(i,j)} \log O(x) = \sum_{s \in path(i,j)} \log \left( \frac{1}{O(s)} \right) = -\log \prod_{x \in path(i,j)} \frac{1}{O(x)} = -\log r_{ij}
\]

Fig. 1. Logarithmic distances in the crawling tree.
Similarly, \( d_{ia} = -\log r_{ia} \) thus when \( d_{ij} < d_{ia} \), we will have \( r_{ja} > r_{ka} \).

Therefore, if the logarithmic distance between \( i \) and \( j \) is less than that 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 \( k \).

If we use Eq. (2) as our selection criteria, after passing several iterations, the values of \( \log(O(i)) \) and \( d_i \) are not comparable and almost the effect of current link's weight will be lost. Thus using Eq. (2), we propose the following formula which is similar to the reinforcement learning algorithm [14] to compute the distance of page \( j \) from the root \( i \) links to \( j \).

\[
\begin{align*}
    d_j &= \log(O(i)) + \gamma \cdot d_j \\
    0 &\leq \gamma \leq 1
\end{align*}
\] (3)

The discount factor \( \gamma \) is used to regulate the effects of the distance of pages in the path leading to page \( j \) on the distance of page \( j \). For example, if we have a path \( k \rightarrow l \rightarrow i \rightarrow j \), then the effect of the distance of \( k \) on \( j \) is regulated with a \( \gamma^3 \) factor. Eq. (4) shows the main formula of FICA which is based on reinforcement learning.

\[
\begin{align*}
    d_{i \
    j} &= (1-\alpha) \cdot d_{ij} + \alpha \cdot (\log(O(i)) + \gamma \cdot d_j) \\
    i &\in B(j), 0 < \alpha \leq 1, 0 \leq \gamma \leq 1
\end{align*}
\] (4)

where \( \alpha \) is the learning rate and \( \log(O(i)) \) is the instantaneous punishment the crawler receives in transition from \( i \) to \( j \). The old distances, \( d_{ij} \) and \( d_j \) show the distance values of pages \( j \) and \( i \) in time \( t \) respectively and \( d_{i \rightarrow j} \) is the new distance of page \( j \) at time \( t + 1 \). In other words, the distance of page \( j \) at time \( t + 1 \) is dependent on its previous distance, its father’s distance \( d_j \) and \( \log(O(i)) \), the instantaneous punishment from selection of page \( j \) by the user. In the environment of the web, our aim is to decrease the sum of punishments received by the crawler. We model the learning rate, \( \alpha \) as in Eq. (5) in which \( t \) shows time or the iteration number and \( \beta \) is a static value to control regularity (see Section 5). Experiments (discussed in the next section) reveal that if the learning rate \( \alpha \) is adjusted properly (it must be slowly decreased) we achieve nearly optimum results and high throughput very fast. When the algorithm starts, we don’t have the distance of any page so we initially set \( \alpha \) to one and, then, decrease it asymptotically to zero. It is notable that we should set alpha such that when it is close to zero we navigated almost the whole of the web graph and there are not any uncrawled pages in the queue. If alpha reaches zero too early the throughput of the algorithm will be decreased.

\[
\alpha = e^\delta t
\] (5)

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. So \( d_j \) is the total punishment (expected) that the agent receives from selecting page \( j \).

As we noted before, each page is retrieved only once. Thus our method is not completely mapped to reinforcement learning (RL) for all pages. But, RL is suitable for uncrawled pages, because their distances will be updated and changed by visiting their links in other pages. In the dynamic version of FICA (Section 7), a page may be crawled many times so it is completely a RL problem.

A major contribution of our method is modelling the random 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 \( \alpha = 1 \), as it visits more pages, it slowly learns from the environment (\( \alpha \) decreases) and selects future pages better than before. Obviously, its learning rate is high at first and decreases to zero through time.

Eq. (4) 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 \( i \) (Fig. 2). 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 dis-
distance value for page \( j \). Thus, the distance formula changes to Eq. (6) which is based on Q-learning [14].

\[
d_{j+1} = (1 - \alpha) * d_j + \alpha * \min_i (\log (O(i)) + \gamma * d_i)
\]

Also we can use other functions such as simple and weighted average to compute the new distance of page \( j \). Overall, we are going to discover and follow the shortest paths to find low distance web pages first. In other words our miracle is to minimize the \( \sum_{i \in C} d_i \) value where \( C \) is the set of currently crawled web pages.

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

\[
D \leftarrow (\infty, \infty, \infty, \hat{D})
\]

**Input:** starting_url \( K=250,000 \), \( t=0 \), \( \beta=0.1 \), \( \text{Size}=0 \), \( \text{distance}=0 \);
\[ \text{enqueue}((\text{URL-queue}, \text{starting_url}, \text{distance})) \]
While (not empty(\text{URL-queue}))
\[ (\text{url}, \text{distance}) = \text{dequeue}((\text{URL-queue}) \]
\[ \text{Size} = \text{Size} \div K \]
\[ \alpha = \beta^t \]
\[ \text{Distance}=\log(0(\text{url}))+\gamma*\text{distance} \]
\[ \text{crawl_page}(\text{url}) \]
for each child \( u \) of \( \text{url} \)
\[ d = (1-\alpha) \times D[u] + \alpha \times \text{distance} \]
if \( d < D[u] \)
\[ \text{enqueue}((\text{URL-queue}, (u, d))) \]
\[ D[u] = d \]
End while

The \( \text{URL-queue} \) 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 \( \text{URL} \), is inserted into the \( \text{URL-queue} \) and the pages are sorted by their distances in ascending order. \( D[u] \) is the distance value of page \( u \). 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 \( \text{crawled_pages} \) variable includes currently crawled pages. The \( \text{crawl_page}(\text{url}) \) function fetches a page from the web and extracts its \( \text{URLs} \). The \( \text{enqueue}((\text{queue}, (\text{element}, d))) \) function inserts an element with distance \( d \) into its position in the priority queue (elements are sorted according to their distances). \( \text{Dequeue}((\text{queue})) \) removes the element at the beginning of the priority queue and its distance and returns them.

The number of iterations or \( t \) is incremented after crawling every \( K \) (250,000) pages. The \( \beta \) value must be set properly for learning rate justification. We found that for \( K=250,000 \), \( \beta = 0.1 \) is suitable.

The variable \( \text{distance} \) is a temporary variable which we use to keep the distance of each parent node when we remove it from the priority queue to compute the distance of its child nodes from distance of the parent.

Upon finding a new uncrawled page, we insert it into the \( \text{URL-queue} \) with its received distance if it is not in the \( \text{URL-queue} \) 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 content faster.

4. Experimental results

For the evaluation of FICA, we used UK’s web [16] which includes over 18 million web pages that have urls ending with .uk. Our goal was to compare FICA with other crawling algorithms and see 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.
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).

FICA uses Algorithm 1 for crawling. Initially, we start crawling the web with every algorithm with some starting URLs. In our experiment we randomly select 1000 pages as starting URLs (roots or seeds). Because $g$ is less than 1 in FICA, dependency of it on the root selection is very low. For example, if page $i$ is / links farther away than the root page ($l$ clicks), then the distance effect of the root on this page is a factor of $g^l$. It is notable contrary to current algorithm, if we run the FICA algorithm many times (Section 7); then $d_i$ will be the average shortest path between from all pages to $i$ not only the root. In other words after some iterations the distance vector is independent of the initial seeds.

However, like TrustRank [27] we can use some trusted pages and pages with a large number of out-links as seeds. With this condition we will discover more good pages and probability of finding the unsuitable pages will be very low.

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 times¹). 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 depicted 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

```c
Input: starting_url, K=250,000  
enqueue(URL_queue, starting_url)  
while (not empty(URL_queue))  
crawl_page(url)  
for each child u of url  
if(u $\notin$ URL_queue and u $\notin$ 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
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:
     
     $$PR\ [u] = \frac{(1 - 0.85)}{n} + 0.85 \sum_{p_r \in B[u]} \frac{PR\ [p_r]}{O[p_r]}$$
     
     where $B[u]$ shows list of pages linking to $u$ and $O[p]$ is the number of links in page $p$.  
     - Sort URL_queue by $PR(u)$

The OPIC algorithm runs with 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, 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 the most 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.

We compared the aforementioned algorithms with FICA in Fig. 3 for 18 million web pages. The damping factor of PageRank, $\gamma$ and $\beta$ were set to 0.85, 0.5 and 0.1 respectively. As Fig. 3 shows, FICA outperforms other algorithms especially in the early stages. For example, when 25% of pages are crawled, FICA finds about 53% of hot pages whereas partial PageRank and OPIC find 44% and 46% respectively. In comparison to PageRank and OPIC, FICA exhibits an 8% increase in throughput with very low complexity while at the same time, it uses less memory and does not need to store the matrix of the web graph.

Modelling a random 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.

¹ 18million/250,000 = 72.
In reinforcement learning algorithms, the learning rate ($\alpha$) plays an important role in the convergence of the system. We compared the throughput of FICA for different learning functions, that is, the exponential (Eq. (5)), $x^{-1}$, power law [15] ($x^{-2.1}$), and linear ($x$) functions where $x = \beta \cdot t$. 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 $\beta$ in the exponential learning rate, and found $\beta = 0.1$ is the best selection. This value causes the learning rate to decrease more slowly and better resembles user behaviour.

Furthermore, we compare FICA’s throughput for different static alphas in Fig. 4 to better clarify the effect of alpha in the learning process. It is because after some iterations $d_i$ will have a big value and the effect of $\log O(i)$ will be lost. So we need an alpha factor that is less than one to regulate this effect. If we compare FICA’s performance when alpha is 0.5 and 0.9, in the first case, at first its throughput is low and with time it learns more from the environment and its throughput increases. But in the second case, at first we have high throughput and finally because the learning rate is very low its throughput decreases.

We conclude that we need to set alpha to a large value initially, and after that we decrease it exponentially via Eq. (5) which results in the highest throughput.

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 the criteria for ranking. That is, pages with lower distances will have higher ranks.

As Algorithm 1 shows, the distance vector $D$ is computed and updated during the crawling process. Also the distance of a crawled page will be updated when the new distance is less than the old one. So the dependency of the selected starting (root) pages on the final distance of each page is low. By running the crawling process in several iterations, this effect will be very low while we achieve better results.

To illustrate the validity of this ranking, we used Kendall’s $\tau$ metric [9] which shows the correlation between two ranked lists. Two identical lists have $\tau = 1$, while two totally uncorrelated lists have $\tau = 0$ and reversed lists have $\tau = -1$.

We calculated this metric for ranked lists produced by each crawling algorithm compared to the ranked list produced by PageRank (benchmark ranking). Because of the complexity of computing Kendall’s metric on the full list of pages, 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 the FICA algorithm, like OPIC, is suitable for ranking 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, a ranking of the pages will be available too.

FICA has a close relationship with the PageRank algorithm. In [33] there is an interpretation of PageRank as the probability that a random surfer ends her walk at a page. If we consider \( r_j \) as the probability of reaching page \( j \) from \( i \) (path probability), then the PageRank of page \( j \) will be the sum of the path probabilities of all surfing lead to page \( j \). FICA tries to discover the pages with the shortest paths (least distance) or the paths with highly probabilities. In FICA we don’t have whole of the web graph and also downloading the pages is costly. Then it should follow shortest paths in the greedy criterion. In other words, FICA tries to find pages with high PageRank.

6. Complexity of FICA

**Lemma 2.** The complexity of the FICA algorithm in the average case is \( O(E \cdot \text{Log}(V)) \).

**Proof.** The average time of insertion or deletion for a binary tree with size \( M \) is \( \text{Log}(M) \). 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 \( O(E \cdot \text{Log}(V)) \).

We use a binary tree for queue management, then, by Lemma 2, the complexity of our algorithm in the average case is reduced to \( O(E \cdot \text{Log}(V)) \). 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. In PageRanked-base crawling algorithms we run PageRank \( V/K \) times (Algorithm 2) and in each step \( V/K \) nodes will be added to the number of available nodes (visited nodes). Therefore, in the first run of PageRank we have \( V/K \) nodes, in second run we have \( 2V/K \) nodes, in third run we have \( 3V/K \) nodes and finally in the last run we have \( KV/K = V \) nodes. Assume that with adding \( V/K \) nodes to the available nodes \( E/K \) edges will be added to the available edges. With this assumption the complexity of crawling algorithm based on PageRank is of \( O(100^*E/K + 100^*2E/K + 100^*3E/K + \cdots + 100E) = O(100*(E/K) \cdot (K*(K+1))/2) = O(50 \cdot E \cdot (K+1)) \).

Thus, FICA is faster than crawling methods based on PageRank by a factor of \( (50^*(K+1))/\text{Log}(V) \). 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 \( O(100050^*E) \) while FICA’s complexity is \( O(10^5*E) \). This estimation is based on a best case assumption. We now that in reality with adding \( V/K \) nodes to the available nodes, more than \( E/K \) edges will be added to the available edges, so we can say that FICA is faster than PageRank-based crawling methods by a factor of even greater than \( (50^*(K+1))/\text{Log}(V) \).

As far as we can imagine, the most time consuming operation of a primitive crawler is the url download time. 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.

The number of crawled pages in both PageRank-based crawling algorithms and FICA is equal. Thus we can ignore the url download time when we want to compare the complexity of FICA with PageRank-based crawling algorithms. However, the computation time of FICA will be distributed among url download times. In fact in FICA the computation time is parallel with url download time and we can update the distance of pages and download the pages concurrently. But in PageRank-based crawling algorithms we cannot have such a parallelism and we should compute the PageRank at the end of each step.

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 run, we will also have relatively acceptable ranks of all pages (see Section 5 for details).
7. Adaptive FICA

The web graph changes dynamically and, consequently, pages and links between them are added and removed frequently. It is shown that new pages and links are created at rates of 8% and 25% every week, respectively [18]. Considering that the crawling process takes a fair amount of time to complete, changes in the structure of the web graph during the crawling process are inevitable. Thus, an adaptive crawling algorithm that can adjust to these changes is necessary. Thus we should use Eq. (4) in another way for adaptation to the web. As search engines crawl the whole or part of the web in each run, we have to set alpha to the appropriate value (not zero or 1) in each iteration to use the knowledge from previous crawling. So we initialize alpha to a static value (for example 0.5) in each run and decrease it exponentially to zero. This value is dependent on the time it takes to crawl and the period between two situational runs.

In this version, each page might be retrieved many times while in the non-adaptive version (Eq. (4)) each page is retrieved only once. Contrary to the previous version, this one is completely mapped to the Reinforcement Learning problem.

In Algorithm 1, we have used a greedy mechanism for crawling. In other words, some remained pages with lower distances were selected in crawling. Thus, we only use exploitation and we don’t have any exploration. It is important to have a policy between exploitation and exploration dilemma. Figure 5 shows a global view of the crawling process. In every stage of crawling we have to select some pages for crawling. We have three types of candidates (i) the uncrawled pages with low distance, (ii) uncrawled pages which currently have high distance and (iii) currently crawled pages in which content may have been changed during the crawling process. Because both the web content and structure are dynamic, this process will iterate for ever by a suitable policy.

Furthermore, in our proposed algorithm (Algorithm 1), the relationship between Reinforcement Learning and the crawling process is pretty weak because the crawler never takes the same action twice. But in the dynamic version of our algorithm we will run the crawling process in many iterations and a page may be retrieved from the web many times. We use the knowledge form the past to have a better crawling process and also find high quality pages faster.

Because the evaluation of dynamic FICA is difficult, we have not compared the adaptive version of FICA with other algorithms. For evaluation we need some snapshots of a portion of the web and we are currently working on it. But we believe the results will be interesting and will report them in future publications.

8. 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 random user’s 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 previous algorithms. Meanwhile, page importance is computed during the crawling process. In other words, after the crawling process we will have an acceptable ranking of pages too.

FICA is fast and easy to implement with low time complexity, around $O(E*\log V)$ 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 has higher throughput, which is a performance metric 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, building on the idea used in FICA, we have proposed a ranking algorithm...
called DistanceRank [17] that outperforms other ranking algorithms. Evaluation of FICA for larger graphs and evaluation of the adaptive version of FICA that adjusts dynamically with changes in the web graph remain as future work.

Acknowledgement

This work has been partially supported by Iran Telecommunication Research Center (ITRC). The authors would like to thank IEEE/WIC’s reviewers and Mohammad Keyhani for valuable comments.

References