Appl Intell (2012) 37:586–601
DOI 10.1007/s10489-012-0351-2

An adaptive focused Web crawling algorithm based on learning automata

Javad Akbari Torkestani

Published online: 29 April 2012

Abstract The recent years have witnessed the birth and explosive growth of the Web. The exponential growth of the Web has made it into a huge source of information wherein finding a document without an efficient search engine is unimaginable. Web crawling has become an important aspect of the Web search on which the performance of the search engines is strongly dependent. Focused Web crawlers try to focus the crawling process on the topic-relevant Web documents. Topic oriented crawlers are widely used in domain-specific Web search portals and personalized search tools. This paper designs a decentralized learning automata-based focused Web crawler. Taking advantage of learning automata, the proposed crawler learns the most relevant URLs and the promising paths leading to the target on-topic documents. It can effectively adapt its configuration to the Web dynamics. This crawler is expected to have a higher precision rate because of construction a small Web graph of only on-topic documents. Based on the Martingale theorem, the convergence of the proposed algorithm is proved. To show the performance of the proposed crawler, extensive simulation experiments are conducted. The obtained results show the superiority of the proposed crawler over several existing methods in terms of precision, recall, and running time. The t-test is used to verify the statistical significance of the precision results of the proposed crawler.

Keywords Web crawling · Focused Web crawler · Search engine · Learning automata

J. Akbari Torkestani (✉)
Young Researchers Club, Arak Branch, Islamic Azad University, Arak, Iran
e-mail: j-akbari@iau-arak.ac.ir

1 Introduction

Due to the huge amount of information available on the ever-growing World Wide Web, insufficient and vague user queries, and use of the same query by different users for different aims, the information retrieval (IR) process in the Web is remarkably hard and deals with a huge amount of uncertainty and doubt [34, 35]. Under such circumstances, designing an efficient IR tool by which the most relevant results are provided is of great importance. Search engines are easy-to-use IR tools that find and rank the most relevant documents based solely on the simple keywords [33, 36]. A search engine is generally made of three main parts: crawler, indexer and query processor. A web crawler (also known as Robot or Spider) assembles the web content locally [6]. Crawler starts at an initial (or seed) URL (short for uniform resource locator), visits and downloads the page, indexes it and follows the hyperlinks within the page to reach the other pages. This process continues until a predefined number of pages are visited or no more relevant pages can be found. Generally speaking, a web crawling system has two main phases: the first one is determination of the seed URLs, and the latter one is the traversing the hyperlinks and downloading the web pages [7]. A focused web crawler is designed to collect the relevant topic-specific Web documents requested by individuals or organizations and can be used in personalized web portals and search engines [6, 8]. Generic web crawlers retrieve the massive numbers of web pages regardless of the topic relevancy, while the focused web crawlers try to traverse the only hyperlinks leading to more relevant Web documents. In [6], focused crawlers are classified as classic crawlers [14], semantic crawlers [3, 15, 16], and learning crawlers [4, 5, 7, 8, 17, 18]. The following surveys the recent literature on focused Web crawling.

In [3], Jung proposed a cooperative focused crawling system based on evolution strategy to overcome the difficulties with the contextual IR in semantic web environment. In the proposed model, the semantic fitness of a population is determined by using the similarity-based ontology matching algorithm. Rungsawang and Angkawattanawit [4] proposed a learning topic-specific web crawling technique in which a knowledge base, comprising the information of the previous crawling attempts, is constructed to make the next crawls more efficient and easier. In this method, the crawler learns how to collect the most relevant pages based on the previous experiences stored in the knowledge base. An intelligent focused web crawler called BioCrawler was proposed by Batzios et al. [5] for semantic web. The idea behind BioCrawler is based on the Biotope concept [19] in which the ecosystem rules are applied to a set of autonomous cooperating agents in order to crawl the Web.

Liu et al. [8] designed a Hidden Markov Model (HMM)-based system for prediction of the links leading to the relevant pages in a focused Web crawler. A HMM is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved (hidden) states. The proposed crawling technique that is called LJM is composed of three stages: user data collection, user modeling via sequential pattern learning, and focused crawling. In the first stage, the user browses the Web looking for the relevant pages and indicates if a downloaded page is topic-relevant or not. The Web graph is constructed based on the pages visited by the user. The sequence of the previous observations is used to update the transition probabilities between different states of the Markov chain. In the crawling stage, the prediction for the next page involves only the transition probabilities. The proposed technique captures the hierarchical semantic linkage structure of the topics. To do so, the collected pages are clustered and the structure connecting the clusters is used to learn the page sequences that more likely lead to the relevant pages. The authors showed that the HMM-based crawler outperforms the context-graph crawler and Best-First crawler. However, the proposed model is too user dependent. Formation of the Web graph and the Web clusters form the pages visited during a user browsing session may exclude many relevant pages from the crawling process. LJM is one of the baselines in this paper. Batsakis et al. [6] addressed the issues related to the design and implementation of the focused web crawlers and reviewed the state-of-the-art crawlers relying on Web page content and link information. This paper also proposes new crawlers by taking into account anchor text, page content and text surrounding the links for computation of download priorities in HMM-based and Best-First crawlers. HMM-based with anchor text, page content and centroid similarity (which is hereafter called BPM) is another baseline with which the results of this paper are compared. HMM-based crawler [8] assigns the same priority to different pages associated with the same cluster. This prevents a good decision making within the cluster. To improve the prioritization mechanism, BPM uses another concept called centroid similarity. The centroid similarity for a page is defined as the amount of similarity between the page content and the cluster centroid (see Ref. [6] for more detail). In BPM, the priority of a page is defined as the average of the priority assigned by HMM-based model [8] and the centroid similarity. Therefore, for different pages associated with the same cluster, BPM can generate different priorities depending on their content. Experimental results reported in [6] show the superiority of BPM [6] over LJM [8].

In [7], Zhang and Lu proposed an online semi-supervised clustering approach for topical focused web crawlers based on Q-learning and semi-supervised learning theories. The proposed technique uses the concept of score to select the most topic-related URL to crawl the unvisited URLs. The score is determined based on the fuzzy membership function and the Q-value of the unlabeled URLs. Patel and Schmidt [11] put forward the impact of the structure of the crawled web documents on the performance of a focused or topic-specific crawler. The authors believe that the structural analysis of the Web documents can help the crawlers to more effectively prioritize the URLs and to improve the convergence speed to a topic. Avoiding the off-topic areas is an appealing approach that is used in the crawler proposed by Hsu and Wu [12] to concentrate on the URLs leading to the documents of interest. The proposed method that is called HW is based on the relevancy context graph. A context graph is constructed for each on-topic document. This graph can estimate the distance and the relevancy degree between the downloaded document and the search topic. After construction of the context graph, the topic-specific word distribution in a page is calculated for ranking the priority of the document. The rank of a document is related to the location (relevancy) of the document in the context graph and the distribution of the feature words in the document. The relevance degree of a document located at layer n of the context graph is α^n, where $\alpha \leq 1$. Simulation results given in [12] show that HW can effectively estimate the proper order of unvisited pages and performs better than pure context-graph and breadth-first crawlers. HW [12] is the third baseline in this paper.

In this paper, a decentralized random algorithm is proposed for focused Web crawling based on the learning automata theory. In the proposed method, a network of learning automata is formed by assignment of a learning automaton to each web document. Each automaton randomly chooses the hyperlinks (URLs) of its corresponding document as its actions. The crawling process starts at the seed URL and the document associated with the selected action continues the crawling process at each stage. For each downloaded document, the similarity between the search topic and the document is estimated. The selected URL is rewarded if the similarity of the document it leads to is the greatest similarity seen so far. Otherwise it is penalized. The crawling process is repeated until no more documents remain or a given number of documents is harvested. For each document, the crawling process converges to the most topic-relevant URLs that minimize the penalty rate. That is, the most relevant documents are selected with a higher probability. For the next crawls, the proposed crawling method learns how to harvest the Web so that the off-topic areas are excluded and the most promising paths leading to the most relevant documents are visited first. The proposed algorithm adaptively updates its configuration according to the Web dynamic. It is proved that by a proper choice of the learning rate of the learning automat-based crawler, it selects the most topic-relevant URL of each Web document with a probability close enough to one. The performance of the proposed method is measured through simulation experiment. The results of the proposed focused crawler is compared with those of BPM [6], LJM [8] and HW [12], and the obtained results show that it outperforms the others in terms of precision, recall, and time complexity. The t-test is used to verify the statistical significance of the precision results of the proposed crawler.

The rest of the paper is organized as follows. The next section of this paper briefly reviews the learning automata theory. In Sect. 3, a learning automata-based focused crawler is presented. Section 4 represents the convergence proof of the proposed crawling algorithm. Section 5 shows the performance of the proposed algorithm through simulation experiments and comparison with several existing methods. Finally, Sect. 6 concludes the paper.

2 Learning automata theory

A learning automaton [22, 23] is an adaptive decision-making unit that improves its performance by learning how to choose the optimal action from a finite set of allowed actions through repeated interactions with a random environment. The action is chosen at random based on a probability distribution kept over the action-set and at each instant the given action is served as the input to the random environment. The environment responds the taken action in turn with a reinforcement signal. The action probability vector is updated based on the reinforcement feedback from the environment. The objective of a learning automaton is to find the optimal action from the action-set so that the average penalty received from the environment is minimized. Learning automata have been found to be useful in systems where incomplete information about the environment exists [24]. Learning automata are also proved to perform well in complex, dynamic and random environments with a large amount of uncertainties. A group of learning automata can cooperate to cope with many hard-to-solve problems. To name just a few, learning automata have a wide variety of applications in combinatorial optimization problems [9, 25, 37], computer networks [1, 2, 13, 26, 38–40], queuing theory [27], signal processing [28], information retrieval [29], adaptive control [30], Grid computing [41], Web engineering [10], and pattern recognition [31].

The environment can be described by a triple $\{\alpha, \beta, c\}$, where $\alpha \equiv \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ represents the finite set of the inputs, $\beta \equiv \{\beta_1, \beta_2, \ldots, \beta_m\}$ denotes the set of the values that can be taken by the reinforcement signal, and $c \equiv \{c_1, c_2, \ldots, c_r\}$ denotes the set of the penalty probabilities, where the element c_i is associated with the given action α_i. If the penalty probabilities are constant, the random environment is said to be a stationary random environment, and if they vary with time, the environment is called a non stationary environment. The environments depending on the nature of the reinforcement signal $\underline{\beta}$ can be classified into P-model, Q-model and S-model. The environments in which the reinforcement signal can only take two binary values 0 and 1 are referred to as P-model environments. Another class of the environment allows a finite number of the values in the interval $[0, 1]$ can be taken by the reinforcement signal. Such an environment is referred to as Q-model environment. In S-model environments, the reinforcement signal lies in the interval $[a, b]$.

Learning automata can be classified into two main families [22]: fixed structure learning automata and variable structure learning automata. Variable structure learning automata are represented by a triple $\langle \underline{\beta}\underline{\alpha}, L\rangle$, where $\underline{\beta}$ is the set of inputs, $\underline{\alpha}$ is the set of actions, and L is learning algorithm. The learning algorithm is a recurrence relation which is used to modify the action probability vector. Let $\alpha_i(k) \in \underline{\alpha}$ and $\underline{p}(k)$ denote the action selected by learning automaton and the probability vector defined over the action set at instant k, respectively. Let a and b denote the reward and penalty parameters and determine the amount of increases and decreases of the action probabilities, respectively. Let r be the number of actions that can be taken by learning automaton. At each instant k, the action probability vector $\underline{p}(k)$ is updated by the linear learning algorithm given in (1), if the selected action $\alpha_i(k)$ is rewarded by the random environment, and it is updated as given in (2) if the taken action is penalized.

$$p_j(k+1) = \begin{cases} p_j(k) + a[1 - p_j(k)] & j = i \\ (1-a)p_j(k) & \forall j \neq i \end{cases} \tag{1}$$

$$p_j(k+1) = \begin{cases} (1-b)p_j(k) & j = i \\ (\frac{b}{r-1}) + (1-b)p_j(k) & \forall j \neq i \end{cases} \tag{2}$$

If $a = b$, the recurrence equations (1) and (2) are called linear reward-penalty (L_{R-P}) algorithm, if $a \gg b$ the given equations are called linear reward-ϵpenalty ($L_{R-\epsilon P}$), and finally if $b=0$ they are called linear reward-Inaction (L_{R-I}). In the latter case, the action probability vectors remain unchanged when the taken action is penalized by the environment.

2.1 Variable action-set learning automata

A variable action-set learning automaton is an automaton in which the number of actions available at each instant changes with time. It has been shown in [32] that a learning automaton with a changing number of actions is absolutely expedient and also ϵ-optimal, when the reinforcement scheme is L_{R-I}. Such an automaton has a finite set of n actions, $\underline{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$. $A = \{A_1, A_2, \ldots, A_m\}$ denotes the set of action subsets and $A(k) \subseteq \alpha$ is the subset of all the actions can be chosen by the learning automaton, at each instant k. The selection of the particular action subsets is randomly made by an external agency according to the probability distribution $\Psi(k) = \{\Psi_1(k), \Psi_2(k), \ldots, \Psi_m(k)\}$ defined over the possible subsets of the actions, where $\Psi_i(k) = \text{prob}[A(k) = A_i | A_i \in A, 1 \leq i \leq 2^n - 1]$.

$\hat{p}_i(k) = \text{prob}[\alpha(k) = \alpha_i | A(k), \alpha_i \in A(k)]$ denotes the probability of choosing action α_i, conditioned on the event that the action subset $A(k)$ has already been selected and $\alpha_i \in A(k)$ too. The scaled probability $\hat{p}_i(k)$ is defined as

$$\hat{p}_i(k) = \frac{p_i(k)}{K(k)} \tag{3}$$

where $K(k) = \sum_{\alpha_i \in A(k)} p_i(k)$ is the sum of the probabilities of the actions in subset $A(k)$, and $p_i(k) = \text{prob}[\alpha(k) = \alpha_i]$.

The procedure of choosing an action and updating the action probabilities in a variable action-set learning automaton can be described as follows. Let $A(k)$ be the action subset selected at instant n. Before choosing an action, the probabilities of all the actions in the selected subset are scaled as defined in (3). The automaton then randomly selects one of its possible actions according to the scaled action probability vector $\hat{p}(k)$. Depending on the response received from the environment, the learning automaton updates its scaled action probability vector. Note that the probability of the available actions is only updated. Finally, the probability vector of the actions of the chosen subset is rescaled as $p_i(k+1) = \hat{p}_i(k+1) \cdot K(k)$, for all $\alpha_i \in A(k)$. The absolute expediency and ε-optimality of the method described above have been proved in [32].

3 Learning automata-based focused crawling

The topic-specific or focused web crawling deals with the collection and indexing the relevant Web documents of interested topics. Due to the huge amount of relevant documents available on the Web, finding a proper order to crawl the Web documents so that the most topic-relevant Web documents are downloaded in the first few visits is one of the most challenging issues that has received the attention of many Web researchers during the last decade. In this paper, a decentralized algorithm called LAFC is proposed for focused Web crawling in which taking advantage of learning automata an order is determined to optimally arrange the crawls.

In the proposed method, the Web is modeled as a connected directed graph with the web documents as the graph nodes and the hyperlinks (URLs) as the graph edges. The Web graph is initially constructed during the first crawling and updated by the next crawls. Let $G\langle D, U\rangle$ be a directed graph denoting the Web structure, where $D = \{d_1, d_2, \ldots, d_n\}$ denotes the set of web documents (or Web graph nodes) and $U \subseteq D \times D$ denotes the set of URLs connecting the documents (or Web graph edges). Let $u_{(i,j)} \in U$ denotes the hyperlink appeared in web document d_i leading to the web document d_j. As the crawling process progresses, the Web graph is formed as follows. Seed URL is defined as the starting link that points to the first web document. The document that is pointed by the seed URL is downloaded and denoted as the first crawled Web document. Normally, this document may include a number of hyperlinks that connect this document to the other Web documents. For each hyperlink of the document, a directed connection is established between the current document and the Web document that the hyperlink leads to. Creation and expansion of the Web graph is directly dependent on the hyperlink that is selected to crawl the other Web documents. Since it is attempted to select the most topic relevant URLs, the Web graph will include the only relevant documents while it is as small as possible. Figure 1 shows a sample Web graph that is constructed during the crawling process.

3.1 Configuration of learning automata

In the proposed crawling method, as the Web graph is constructed, a network of learning automata isomorphic to the Web graph is formed as follows. Each crawled web document d_i is associated with a learning automaton A_i. The hyperlinks of each document d_i constitute the action-set of the corresponding learning automaton A_i. That is, the action-set of automaton A_i is defined as $\underline{\alpha_i} = \{\alpha_i^j | \forall u_{(i,j)} \in U\}$. Selection of action α_i^j by learning automaton A_i means that document d_i selects Web document d_j to traverse. As mentioned earlier, each learning automaton chooses its actions on the basis of an action probability vector. Let $\underline{p_i} = \{p_i^j | \forall \alpha_i^j \in \alpha_i\}$ denotes the action probability vector associated with learning automaton A_i, where p_i^j is the choice probability of action $\alpha_i^j \in \alpha_i$. In other words, p_i^j represents the probability with which URL $u_{(i,j)}$ pointing to document d_j is chosen among all URLs in Web document d_i. As a web document

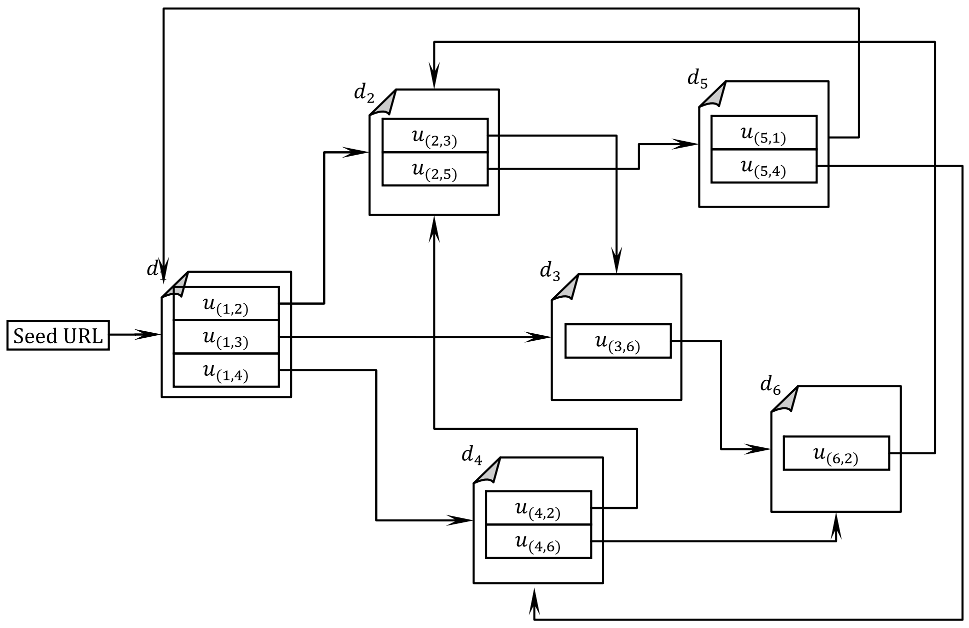

Fig. 1 A sample Web graph

d_i is visited for the first time, its action-set is formed and initial action probability vector is defined. Since the relevancy degree of the URLs of a document (with respect to the topic-specific) is a priori unknown, the choice probability of all URLs is initially set to $\frac{1}{|u_{(i)}|}$, where $u_{(i)}$ denotes the set of URLs (outgoing links) of document d_i.

3.2 Crawling process

The topic description and initial (seed) URL are submitted to the focused crawler as input parameters. Topic description v_t is defined as a m-dimensional vector composed of m topic keywords. Seed URL is pushed into a stack structure that is called stack $\mathcal{S}$. Such a stack-based buffering technique lets the proposed algorithm to crawl the Web in a prioritized depth first order. Web crawler starts at a given seed URL. The Web document to which the seed URL leads is downloaded. This document is denoted as d_i and called current Web document. The content of the current Web document d_i is lexically processed and the vector space model (VSM) [20] of the document is formed. To calculate the topic-document similarity, the frequency of the terms in the VSM (i.e., the number of appearances of the term in document) is measured. Let v_d be the m-dimensional term vector of document d. The hyperlinks of the current Web document d_i are extracted to form the action set of learning automaton A_i.

If a learning automaton has been already associated with the Web document d_i, the action-set and action probability vector of the corresponding learning automaton are updated if it is required. Otherwise, a learning automaton A_i is associated with the current Web document. The action-set of the current automaton A_i is formed and its action probability vector is initialized. The random environment in which the learning automata operate is P-model, and the reinforcement scheme by which the internal state of the learning automata is updated is L_{R-I}. To avoid revisiting the Web documents, the actions corresponding to the documents that have been already visited must be temporarily removed from the action-set of the current automaton (i.e., $\underline{\alpha_i}$). To do so, the learning automaton that is assigned to each Web document is of variable action-set type as described in Sect. 2.1.

Learning automaton A_i randomly chooses one of its actions according to the action probability vector, if any (i.e., if $\underline{\alpha_i} \neq \emptyset$). Let us assume that learning automaton A_i chooses action α_i^j (or URL $u_{(i,j)}$). The Web document corresponding to the selected action (i.e., document d_j) is downloaded, visited, and set as the current Web document. The action that is selected by automaton A_i is temporarily removed from its action-set α_i as described in Sect. 2.1 on variable action-set learning automata. The similarity between the topic vector v_t and the term vector of the current Web document d_j (i.e., v_{d_j}) is computed by (4).

$$\mathrm{Sim}(v_t, v_{d_j}) = \frac{\overrightarrow{v_t} \cdot \overrightarrow{v_{d_j}}}{|\overrightarrow{v_t}| \times |\overrightarrow{v_{d_j}}|} = \frac{\sum_{i=1}^{m} \delta_t^i \delta_{d_j}^i}{\sqrt{\sum_{i=1}^{m} {\delta_t^i}^2 \sum_{i=1}^{m} {\delta_{d_j}^i}^2}} \tag{4}$$

where δ_t^i and $\delta_{d_j}^i$ denote the frequency of term i (for $i \in \{1, 2, \ldots, m\}$) in topic vector v_t and current document d_j, respectively. The term frequency (tf) of a given term i in Web document q is defined as

$$\delta_q^i = \frac{f_i}{\max_i f_i} \tag{5}$$

where f_i denotes the frequency of term i in document q and $\max_i f_i$ denotes the maximum frequency of all terms (of v_q) in this document.

If $\mathrm{Sim}(v_t, v_{d_j})$ is greater than or equal to dynamic similarity threshold τ_i, then the selected action α_i^j is rewarded according to (1) and dynamic similarity threshold τ_i is updated to $\mathrm{Sim}(v_t, v_{d_j})$. Learning automaton A_i penalizes the chosen action according to (2) otherwise. For each Web document d_i, similarity threshold τ_i is defined as the maximum similarity between the search topic and the Web documents to which the hyperlinks of d_i have led so far. That is, dynamic similarity threshold τ_i equals to the maximum similarity seen so far between the URLs of Web document d_i and topic vector v_t as shown in (6). For each Web document d_i, the initial value of τ_i is set to zero. Such an updating rule leads the learning automaton to choose the URL having the maximum similarity with the search topic.

$$\tau_i = \max_{u_{(i,j)} \in u_{(i)}} \mathrm{Sim}(v_t, v_{d_j}) \tag{6}$$

After updating the action probability vector, the proposed algorithm checks to see if the crawled path (the chain of visited documents) is suitable to continue or not. A path is worthwhile to continue crawling until the similarity of the last visited Web document with the topic does not fall below a specified threshold. This part of the proposed crawling mechanism aims at controlling the expansion of the Web graph in such a way that the Web documents having a desired level of relevancy are only crawled. To do so, the similarity of the last visited document d_j (i.e., $\mathrm{Sim}(v_t, v_{d_j})$) is compared with a pre-specified threshold that is called control threshold $\mathcal{C}$. If $\mathrm{Sim}(v_t, v_{d_j})$ is greater than or equal to threshold $\mathcal{C}$, URL $u_{(i,j)}$ is pushed on top of stack $\mathcal{S}$ and the crawling process is continued by the current Web document d_j. In this case, Web document d_j repeats the same operations as document d_i did. That is, a learning automaton is configured for the current document, the automaton chooses one of its action, the similarity is computed, probability vector is updated and the control threshold is verified. Otherwise (i.e., if $\mathrm{Sim}(v_t, v_{d_j}) \ngeq \mathcal{C}$), document d_j is not a promising Web document to traverse and so the crawling process must be resumed from the last URL pushed in stack $\mathcal{S}$. In this case, the last URL is popped from the top of stack $\mathcal{S}$, and the Web document at which this URL leads continues the crawling as document d_i.

As mentioned earlier, the actions corresponding to the documents that have been already visited are disabled to avoid revisiting the Web documents. This may lead us to a learning automaton with empty action-set (i.e., if $\underline{\alpha_i} = \emptyset$). This case usually may occur at the end of the crawling process, when the neighbors of a document in the Web graph have been already visited. In this case, like the case where the similarity rate of a document is not satisfiable, the crawling process is resumed from top URL in stack $\mathcal{S}$.

This process continues until end condition ϵ is met. End condition could be visiting a predefined number of Web documents or not remaining more unvisited Web documents (i.e., no automaton with non-empty action-set). At the end of the crawling process, all learning automata must enable the disabled actions again for the further crawling. By the proposed crawling algorithm, the most related URLs are selected with a higher probability and so the most topic-relevant documents are visited first. This performs well (by downloading the most relevant documents) in cases where a limited number of documents are visited.

3.3 Web dynamic maintenance

The content and structure of the Web is continuously changing and the frequency of this change is becoming more and more. This imposes a heavy burden on the Web crawlers for discovering the Web changes to keep them up to date. As mentioned earlier, in the proposed crawling method If a Web document d_i changes, it might be needed to update the configuration of the learning automaton associated with this document.

Let us assume that a new hyperlink $u_{(i,j)}$ is added to the Web document d_i. In this case, when the crawler revisits this document it updates the corresponding learning automaton A_i by addition of a new action α_i^j, i.e., $\underline{\alpha_i} \leftarrow \underline{\alpha_i} + \alpha_i^j$. The choice probability of the new action α_i^j (or URL $u_{(i,j)}$) is initialized to $\frac{1}{|u_{(i)}|}$, where $|u_{(i)}|$ denotes the current number of hyperlinks of Web document d_i. Furthermore, the choice probability of the other actions must be also updated. To do so, for every action $\alpha_i^k \in \underline{\alpha_i}$, choice probability p_i^k changes as

$$p_i^k \leftarrow \frac{|u_{(i)}| - 1}{|u_{(i)}|} \cdot p_i^k \tag{7}$$

When revisiting the document, if the crawler detects that a given hyperlink $u_{(i,j)}$ has been deleted from the Web document d_i, it removes action α_i^j from the action-set of learning automaton A_i. To adjust the action probability vector p_i, the choice probability of the removed action p_i^j must be distributed between the remaining actions proportional to their choice probability values. This is done for each remaining action $\alpha_i^k \in \underline{\alpha_i}$ by

$$p_i^k \leftarrow \left(1 + \frac{p_i^j}{1 - p_i^j}\right) \cdot p_i^k \tag{8}$$

4 Convergence proof

In this section, it is proved that by a proper choice of the learning rate, each learning automaton associated with a Web document converges to its optimal action if it updates its internal state (action probability vector) by the updating rules presented in LAFC. That is, by a proper learning rate, the learning automata-based crawler chooses the most topic-relevant URL of each Web document with a probability close enough to one.

Theorem 1 *Let $p_i^j(k)$ be the choice probability of the most topic-relevant URL $u_{(i,j)}$ belonging to Web document d_i at stage k by learning automaton A_i. If vector $\underline{p_i}(k)$ is updated according to LAFC, then for every error rate $\varepsilon > 0$, there exists a learning rate $a^*(\varepsilon) \in (0,1)$ such that for all $a \in (0, a^*)$, we have*

$$\operatorname{Prob}\Big[\lim_{k\to 8} p_i^j(k) = 1\Big] \geq 1-\varepsilon \tag{9}$$

Proof To prove this theorem, it is initially required to demonstrate that the choice probability $p_i^j(k)$ is a sub-Martingale process for large values of k and the changes in the conditional expectation of $p_i^j(k)$ are always non-negative. Therefore, the following lemma must be proved first.

Lemma 1 *The changes in the conditional expectation of $p_i^j(k)$ are always non-negative, subject to updating vector $\underline{p_i}$ by LAFC. That is, $\Delta p_i^j(k) > 0$.*

Proof Define the conditional expectation of $p_i^j(k)$ as

$$\Delta p_i^j(k) = E\big[p_i^j(k+1)\big|\underline{p_i}(k)\big] - p_i^j(k) \tag{10}$$

$\Delta p_i^j(k)$ is rewritten as

$$\Delta p_i^j(k) = a p_i^j(k) \sum_{l\neq j} p_i^l(k)\big[c_i^l(k) - c_i^j(k)\big] \tag{11}$$

Let S_r^o denotes the interior of S_r, where $S_r = \{\underline{p_i}(k) | 0 \leq p_i^j(k) \leq 1, \sum_l p_i^l(k) = 1\}$. Therefore, we have $p_i^j(k) \in (0,1)$, for all $\underline{p_i} \in S_r^o$. Theorem 1 assumes that action α_i^j is the optimal action (action with the minimum penalty probability) of learning automaton A_i. That is, URL $u_{(i,j)}$ is the most topic-relevant URL of document d_i. Hence, it is concluded that $c_i^l - c_i^j > 0$, where c_i^j is the final value of penalty probability $c_i^j(k)$, where k is large enough. Using weak law of large numbers, it can be proved that $c_i^j(k)$ converges to c_i^j, for large values of k (i.e., $\lim_{k\to\infty} \operatorname{Prob}[|c_i^j - c_i^j(k)| > \varepsilon] \to 0$). Therefore, it is concluded that for large values of k, the right hand side of (11) is composed of nonnegative quantities, and so we have

$$\Delta p_i^j(k) = a p_i^j(k) \sum_{l\neq j} p_i^l(k)\big[c_i^l(k) - c_i^j(k)\big] \geq 0 \tag{12}$$

that completes the proof of Lemma 1. □

Let $\Gamma_i^j(\underline{p_i})$ denotes the convergence probability of LAFC to the desired state with initial probability vector $\underline{p_i}$. That is,

$$\Gamma_i^j(\underline{p_i}) = \operatorname{prob}\big[p_i^j(\infty) = 1\big|\underline{p_i}(0) = \underline{p_i}\big]$$

Let $C(S_r): S_r \to \mathfrak{R}$ be the state space of all real-valued continuously differentiable functions with bounded derivative defined on S_r, where $\mathfrak{R}$ is the real line. If $\psi(\cdot) \in C(S_r)$, the operator U is defined by

$$U\psi(\underline{p_i}) = E\big[\psi\big(\underline{p_i}(k+1)\big)\big|\underline{p_i}(k) = \underline{p_i}\big], \tag{13}$$

where $E[\cdot]$ represents the mathematical expectation. Operator U is linear and preserves the nonnegative functions as the expectation of a nonnegative function remains nonnegative. That is, $U\psi(\underline{p_i}) \geq 0$ for all $q \in S_r$, if $\psi(\underline{p_i}) \geq 0$. If operator U is repeatedly applied n times (for all $n > 1$), we have

$$U^{n-1}\psi(\underline{p_i}) = E\big[\psi\big(\underline{p_i}(k+1)\big)\big|\underline{p_i}(k) = \underline{p_i}\big]$$

Function $\psi(\underline{p_i})$ is called super-regular (or sub-regular) if and only if $\psi(\underline{p_i}) \geq U\psi(\underline{p_i})$ (or $\psi(\underline{p_i}) \leq U\psi(\underline{p_i})$), for all $\underline{p_i} \in S_r$. $\Gamma_i(\underline{p_i})$ is the only continuous solution of equation $U\Gamma_i^j = \Gamma_i^j$ with the following boundary conditions.

$$\begin{aligned} &\Gamma_i^j(e_i^j) = 1 \\ &\Gamma_i^j(e_i^l) = 0; \quad l \neq j \end{aligned} \tag{14}$$

Define

$$\phi_i^j[x, \underline{p_i}] = \frac{e^{-\frac{x p_i^j}{a}} - 1}{e^{-\frac{x}{a}} - 1}$$

where $x > 0$ is to be chosen. $\phi_i^j[x, \underline{p_i}] \in C(S_r)$ and satisfies the boundary conditions above.

In what follows, it is shown that $\phi_i^j[x, \underline{p_i}]$ is a sub-regular function, thus $\phi_i^j[x, \underline{p_i}]$ qualifies as a lower bound on $\Gamma_i^j(e_i)$. Since super and sub-regular functions are closed under addition and multiplication by a positive constant, and if $\phi_i(\cdot)$ is super-regular then $-\phi_i(\cdot)$ is sub-regular, it follows that $\phi_i^j[x, \underline{p_i}]$ is sub-regular if and only if

$$\theta_i^j[x, \underline{p_i}] = e^{-\frac{x p_i^j}{a}} \tag{15}$$

is super-regular. It is now determined the conditions under which $\theta_i^j[x, \underline{p_i}]$ is super-regular. From the definition of operator U given in (13), we have

$$U\theta_i^j[x, \underline{p_i}] = E\big[e^{-\frac{x p_i^j(k+1)}{a}} |\underline{p_i}\big]$$

If $\theta_i^j[x, \underline{p_i}]$ is super-regular, we have

$$U\theta_i^j[x, \underline{p_i}] - \theta_i^j[x, \underline{p_i}] \le \left[e^{-\frac{xp_i^j}{a}} p_i^j r_i^j e^{-x(1-p_i^j)} + e^{-\frac{xp_i^j}{a}} \sum_{l \ne j} p_i^l r_i^l e^{xp_i^j} \right] - e^{-\frac{xp_i^j}{a}}$$

where r_i^j denotes the probability of rewarding action α_i^j. Define $V[u]$ as follows

$$V[u] = \begin{cases} \frac{e^u - 1}{u}; & u \ne 0 \\ 1; & u = 0 \end{cases} \tag{16}$$

$$\begin{aligned} &U\theta_i^j[x, \underline{p_i}] - \theta_i^j[x, \underline{p_i}] \\ &\quad \le -xp_i^j e^{-\frac{xp_i^j}{a}} \Big[(1-p_i^j) r_i^j V[-x(1-p_i^j)] \\ &\qquad - \sum_{l \ne j} p_i^l r_i^l V[xp_i^j] \Big] = -xp_i^j \theta_i^j[x, \underline{p_i}] G_i^j[x, \underline{p_i}] \end{aligned}$$

where $G_i^j[x, \underline{p_i}]$ is defined as

$$(1-p_i^j) r_i^j V[-x(1-p_i^j)] - \sum_{l \ne j} p_i^l r_i^l V[xp_i^j]$$

Therefore, $\theta_i^j[x, \underline{p_i}]$ is super-regular if

$$G_i^j[x, \underline{p_i}] \ge 0 \tag{17}$$

for all $\underline{p_i} \in S_r$. $\theta_i^j[x, \underline{p_i}]$ is super-regular if

$$g_i(x, \underline{p_i}) = \frac{V[-x(1-p_i^j)]}{V[xp_i^j]} \le \frac{\sum_{l \ne j} p_i^l r_i^l}{(1-p_i^j) r_i^j} \tag{18}$$

The right hand side of the inequality (18) consists of the nonnegative terms, so we have

$$\sum_{l \ne j} p_i^l \min_{l \ne j} \frac{r_i^l}{r_i^j} \le \frac{1}{1-p_i^j} \sum_{l \ne j} p_i^l \frac{r_i^l}{r_i^j} \le p_i^l \max_{l \ne j} \frac{r_i^l}{r_i^j}$$

Substituting $\sum_{l \ne j} p_i^l$ by $1 - p_i^j$ in the above inequality, we have

$$\min_{l \ne j} \frac{r_i^l}{r_i^j} \le \frac{\sum_{l \ne j} p_i^l \frac{r_i^l}{r_i^j}}{\sum_{l \ne j} p_i^l} \le \max_{l \ne j} \frac{r_i^l}{r_i^j}$$

From (18), it follows that $\theta_i^j[x, \underline{p_i}]$ is super-regular if we have

$$g_i(x, \underline{p_i}) \ge \max_{l \ne j} \frac{r_i^l}{r_i^j}$$

For further simplification, let employ logarithms as $\Delta(x, \underline{p_i}) = \ln g_i(x, \underline{p_i})$. We have

$$-\int_0^x H'(u)\, du \le \Delta(x, \underline{p_i}) \le -\int_{-x}^0 H'(u)\, du$$

Where $H'(u) = \frac{dH(u)}{du}$ and $H(u) = \ln V(u)$.

We have

$$\frac{1}{V[x]} \le \frac{V[-x(1-p_i^j)]}{V[xp_i^j]} \le V[-x]$$

and

$$\frac{1}{V[x]} = \max_{l \ne j} \frac{r_i^l}{r_i^j} \tag{19}$$

Let x^* denotes the value of x for which (19) is true. There exists a value of $x > 0$ under which (19) is satisfied, if $\frac{r_i^l}{r_i^j}$ is smaller than 1 for all $l \ne j$. By choosing a value $x = x^*$, (19) holds. Consequently (17) is true and $\theta_i^j[x, \underline{p_i}]$ is a super-regular function, thus

$$\phi_i^j[x, \underline{p_i}] = \frac{e^{-\frac{xp_i^j}{a}} - 1}{e^{-\frac{x}{a}} - 1}$$

is a sub-regular function satisfying the boundary conditions (14). If $\psi_i(\cdot) \in C(S_r)$ is sub-regular with $\psi(e_i^j) = 1$ and $\psi(e_i^l) = 0$ for $l \ne j$, then $\psi(\underline{p_i}) \le \Gamma_i^j(\underline{p_i})$ for all $\underline{p_i} \in S_r$. Therefore, it is concluded that

$$\phi_i^j[x, \underline{p_i}] \le \Gamma_i^j(\underline{p_i}) \le 1$$

From the definition of $\phi_i^j[x, \underline{p_i}]$, we see that given any $\varepsilon > 0$ there exists a positive constant $a^* < 1$ such that for all $0 < a \le a^*$, we have

$$1 - \varepsilon \le \phi_i^j[x, \underline{p_i}] \le \Gamma_i^j(\underline{p_i}) \le 1$$

Hence, it is concluded that the choice probability of the most topic-relevant URL $u_{(i,j)}$ converges to 1 for large values of k, and hence the proof of Theorem 1. □

5 Numerical results

In this section, several simulation experiments (in four sets) are conducted to investigate the performance of the proposed focused crawling algorithm. The first set of simulation experiments (Experiment I) is performed to calibrate the proposed algorithm for further experiments by determination of the proper values for learning rate a and control threshold C. The second, third, and forth sets of simulation

Table 1 Search topics and seed URLs

No.	Topic	Seed URL
1	Linux	http://dir.yahoo.com/Computers_and_Internet/Software/Operating_Systems/UNIX/Linux/ http://www.ask.com/web?q=linux&qsrc=0&o=0&l=dir
2	Robotics	http://dir.yahoo.com/Science/Engineering/Mechanical_Engineering/Robotics/ http://www.ask.com/web?q=robotics
3	Java Programming	http://dir.yahoo.com/Computers_and_Internet/Programming_and_Development/Languages/Java/ http://www.ask.com/web?q=java%20programming
4	Asthma	http://dir.yahoo.com/Health/Diseases_and_Conditions/Asthma/ http://www.ask.com/web?q=asthma
5	Olympic Games	http://dir.yahoo.com/Recreation/Sports/Events/International_Games/Olympic_Games/ http://www.ask.com/web?q=olympic+games

experiments (Experiments II, III, and IV) aim at comparing the results of the proposed crawler with those of the following three baselines in terms of precision, recall, and running time: (1) the HMM-based focused crawling algorithm proposed by Liu et al. [8] called LJM, (2) the improved version of the HMM-based crawler called BPM presented by Batsakis et al. [6], and (3) the relevancy context graph-based topic-specific crawler called HW proposed by Hsu and Wu [12]. Precision and recall are the standard measures that are widely used to evaluate the performance of the IR systems. Precision is the proportion of the retrieved documents that are relevant, while recall is the proportion of relevant documents that are retrieved. Precision $\mathcal{P}$ is defined as the ratio of the on-topic crawled documents to the total number of downloads as

$$\mathcal{P} = \frac{\mathcal{N}_R}{\mathcal{N}_T}, \tag{20}$$

where $\mathcal{N}_R$ denotes the number of topic-relevant documents, and $\mathcal{N}_T$ denotes the total number of harvest documents.

Recall is measured as the ratio of the number of relevant crawled documents to the total number of relevant documents on the Web. Recall has always been a difficult measure to calculate because it requires the knowledge of the total number of relevant items (relevant set) in the collection. It becomes more difficult when the collection size grows. In traditional IR systems, the relevant set is restricted to a given data collection or a database at most. However, for the IR tasks in the Web as a huge and ever-growing source of information, the relevant set is too large and generally unknown. Therefore, recall is very hard to measure for the Web queries. In this paper, to make its measurement possible, the relevant set is defined as the set of all relevant documents returned by different crawlers and recall $\mathcal{R}$ as

$$\mathcal{R} = \frac{\mathcal{N}_R}{\mathcal{N}_S}, \tag{21}$$

where $\mathcal{N}_S$ denotes the relevant set for each query topic and seed URL. In this study, a crawled document is topic-relevant if its similarity is greater than 0.75. Learning automata operate in a P-model random environment and learning algorithm by which the internal state of the learning automata is updated is L_{R-I}.

To make the evaluation and comparison of the conducted simulation experiments fair, the same simulation setup and experiment procedure as those applied in [6] are kept. Table 1 shows a list of five common search topics ("Linux", "Robotics", "Java Programming", "Asthma", and "Olympic Games") and the related seed URLs from Yahoo Directory [21] and search results of query submitted to Ask search engine on which the performance of crawlers is evaluated. In these experiments, the end condition of the proposed crawler, ϵ, is defined as downloading a predefined number of Web documents that is set to 10,000 documents in this paper. That is, for each search topic, crawler stops if the number of downloaded documents is 10,000. The seed URLs that are used for each topic are adopted from [6].

5.1 Experiment I

This set of simulation experiments is performed to calibrate the proposed focused crawling algorithm. A parameter that must be initially tuned to improve the performance of the proposed algorithm is control threshold $\mathcal{C}$. By this threshold, algorithm controls that the Web graph includes only the Web documents with a desired level of relevancy. Needless to say, the selection of a proper value of $\mathcal{C}$ significantly improves the precision. However, the total number of harvest Web documents is directly proportional to the control threshold $\mathcal{C}$ and increases as $\mathcal{C}$ becomes larger. Selection of a small control threshold reduces the precision of the algorithm. On the other hand, some topic-relevant documents might be missed if the control threshold is selected very large (close to one).

Table 2 Comparison of precision and running time of the proposed crawler for different values of control threshold with learning rate 0.1

Control threshold	Precision	Running time (min)
0.45	0.1123	623.109
0.50	0.1227	670.861
0.55	0.1342	707.292
0.60	0.1403	765.312
0.65	0.1512	810.412
0.70	0.1642	860.145
0.75	0.1702	900.214
0.80	0.1861	955.092
0.85	0.1872	1207.68
0.90	0.1890	1415.51
0.95	0.1912	1891.41

Table 3 Impact of the learning rate on precision and running time of the proposed crawler where control threshold is set to 0.8

Learning rate	Precision	Running time (min)
0.05	0.2305	2315.37
0.06	0.2287	2092.04
0.07	0.2216	1878.93
0.08	0.2205	1562.17
0.09	0.2189	1023.12
0.10	0.1861	955.092
0.20	0.1675	870.429
0.30	0.1276	621.974
0.40	0.1112	586.236
0.50	0.0732	432.891

To find an appropriate value of the control threshold $\mathcal{C}$, it changes from 0.45 to 0.95, where learning rate is set to 0.1 and the number of downloaded documents is 10,000 for the first search topic given in Table 1. The obtained results are summarized in Table 2. The results given in this table show that a good trade-off between the running time of LAFC and its precision can be made if the control threshold is set to 0.8. It can be seen that for larger values of 0.8 the algorithm is very time consuming and for smaller values the precision considerably reduces.

As mentioned earlier, the performance of a learning automata-based algorithm is strongly dependent on the choice of the learning rate. By the proper choice of the learning rate of the proposed algorithm, a trade-off between the cost of algorithm (e.g., computational cost) and the precision (percentage of relevant documents) of the results can be made. Obviously, the cost of algorithm decreases as the learning rate increases and vice versa. On the other side, the precision increases as the learning rate decreases. Depending on the application nature, different learning rate can be chosen. If an application sacrifices the cost in favor of the solution optimality, small learning rates are preferred, and a larger one can be chosen otherwise. The aim of this experiment is to find the learning rate for which the precision rate (i.e., ratio of the number of topic-relevant documents to the total number of harvest documents) is maximized in an as short as possible running time. To this end, different learning rates ranging from 0.05 to 0.5 are tried, where 10,000 documents are harvested for search topic #1 and the control threshold is set to 0.8. The obtained results are summarized in Table 3. Comparing the results given in Table 3, it can be seen that the maximum precision with respect to the required running time is achieved when the learning rate is set to 0.09. This calibration of algorithm (i.e., learning rate 0.09 and control threshold 0.8) is used in all further simulation experiments.

5.2 Experiment II

In this set of simulation experiments, the performance of the proposed topic-specific Web crawling algorithm is compared with that of BPM [6], LJM [8], and HW [12] in terms of precision (the percentage of crawled topic-relevant documents). Figures 2 through 6 plot the precision of different crawlers against the number of downloaded documents for search topics 1-5. Number of downloads changes from 1000 to 10,000 with increment step 1000. In all Figs. 2–6, it can be seen that the precision of all algorithms reduces as the number of downloads increases. Obviously, this is due to the fact that all algorithms attempt to crawl the most relevant documents first. Therefore, the number of relevant documents (i.e., relevancy degree) decreases as the number of downloads increases.

Figure 2 depicts the results of crawlers for search topic "Linux". Comparing the curves of different crawlers, it is observed that the proposed crawler LAFC significantly outperforms the others. BPM [6] lags behind LAFC, HW [12] is ranked below BPM [6], and LJM [8] has the worst results. The proposed crawler taking advantage of learning automaton learns the most relevant URLs and the promising paths leading to the target on-topic documents, easily and quickly updates its configuration according to the Web dynamic, and crawls the only Web documents having a desired level of relevancy. That is why it is superior to the others. BPM [6] improves the performance of LJM [8] by combination of the hidden Markov model with page content, anchor text, and centroid similarity (relevant pages cluster centroid), and it is expected BPM [6] outperforms LJM [8]. Another reason for the higher precision rate of BPM [6] compared to LJM [8] is that BPM [6] is able to assign different priorities to different pages of the same cluster depending on their relevancy. However, the computational complexity of BPM [6] is longer than that of LJM [8]. Comparing the curves of different crawlers in Fig. 2, it is found that the reduction ratio of

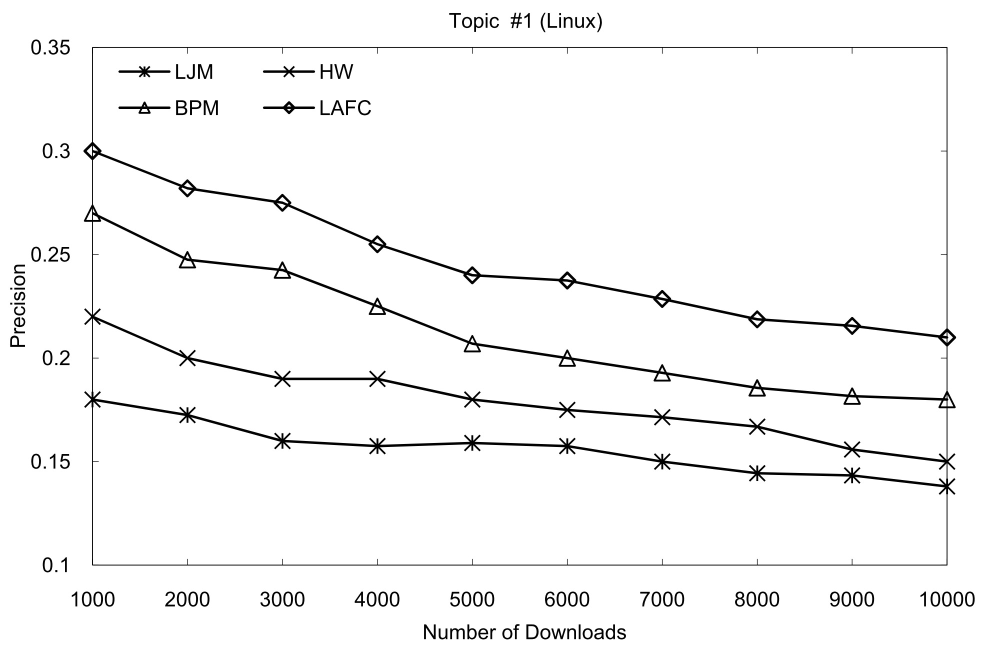

Fig. 2 Precision of crawlers for search topic #1 (Linux)

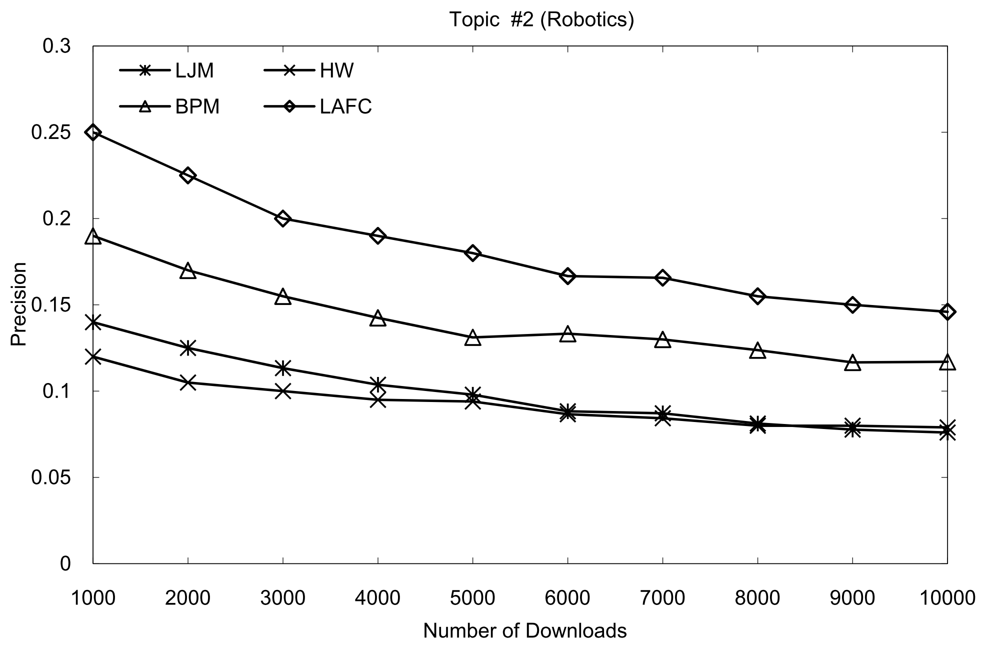

Fig. 3 Precision of crawlers for search topic #2 (Robotics)

precision of LAFC and BPM [6] is slightly higher than that of the others. This could be due to the fact that these crawlers perform better than the others and so harvest a larger number of on-topic documents first.

Figure 3 shows the precision of studied crawlers for topic #2 “Robotics”. It can be seen that LAFC is considerably superior to the others for this search topic too. Depicted curves show that the precision of LJM [8] is slightly more than that of HW [12] specifically for small number of downloaded documents. As the number of downloads increases (larger than 5000), the gap between LJM [8] and HW [12] decreases. The precision curve of BPM [6] lags far behind the proposed crawler for this topic too. Comparing the results shown in Figs. 4, 5 and 6, it can be seen that for topics “Java Programming”, “Asthma”, and “Olympic Games” the proposed crawling technique and LJM [8] perform best and worst, respectively. The results also show the superiority of BPM [6] over HW [12]. Figure 7 represents the overall performance of different crawlers for all considered search topics. This figure summarizes the precision of different crawlers averaged over all five topics given in Table 1. The average results clearly confirm the outperformance of LAFC.

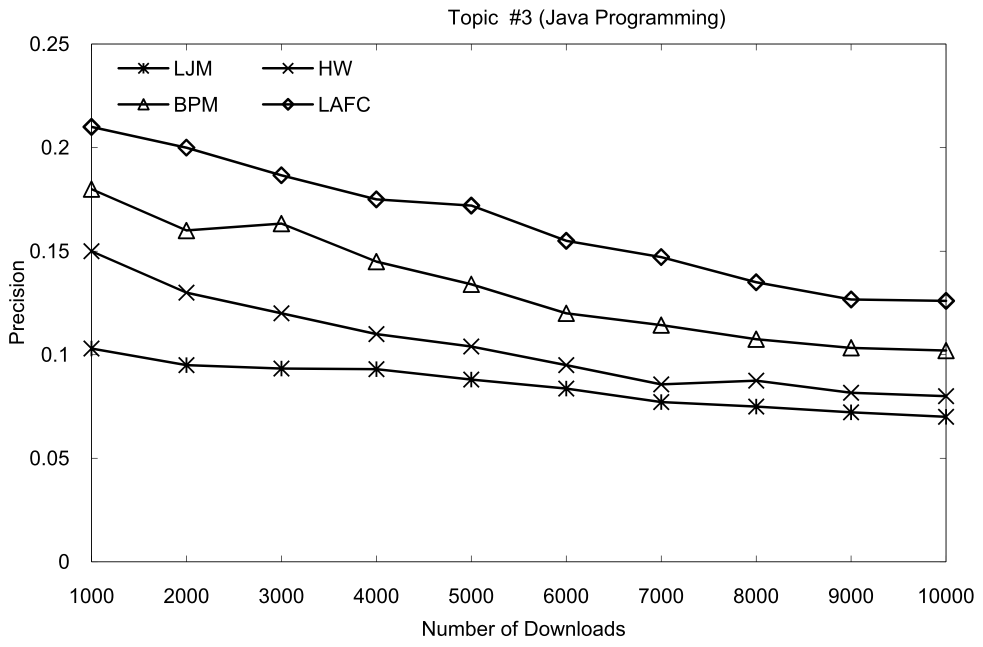

Fig. 4 Precision of crawlers for search topic #3 (Java Programming)

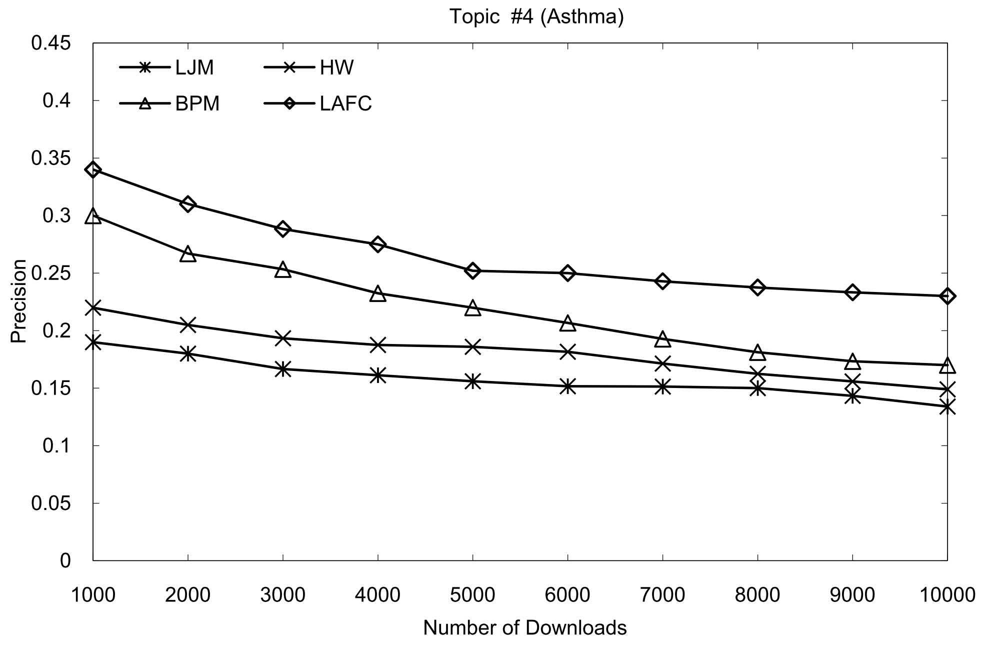

Fig. 5 Precision of crawlers for search topic #4 (Asthma)

To verify the statistical significance of the results of the proposed crawler in comparison with the others, t-test is used for comparing the precision of each pair of crawlers. This test is performed on the average results for all search topics given in Fig. 7 where the number of downloads is set to 10,000. In this test, it is assumed that the difference between each pair of crawlers is statistically significant, if the difference significance is smaller than 0.05. The test results are summarized in Table 4. The first column (Crawler x) of this table includes the list of studied crawlers. The second column shows the mean precision (±standard deviation) of each crawler x. This column shows that LJM [8] has the minimum precision and LAFC has the maximum precision. The third column contains the test results. In this column, the results are given for each pair of crawlers (x, y). Symbol "+" (or "−") appeared in the column labeled as "Performance" indicates that crawler x performs better (or worse) than crawler y in terms of precision. This conclusion is drawn on the basis of the mean precision $(x - y)$ and difference significance. For instance, mean precision $(x - y)$ of 0.0388 and difference significance of 3.45E−06 show that BPM [6] outperforms LJM [8]. Based on the test re-

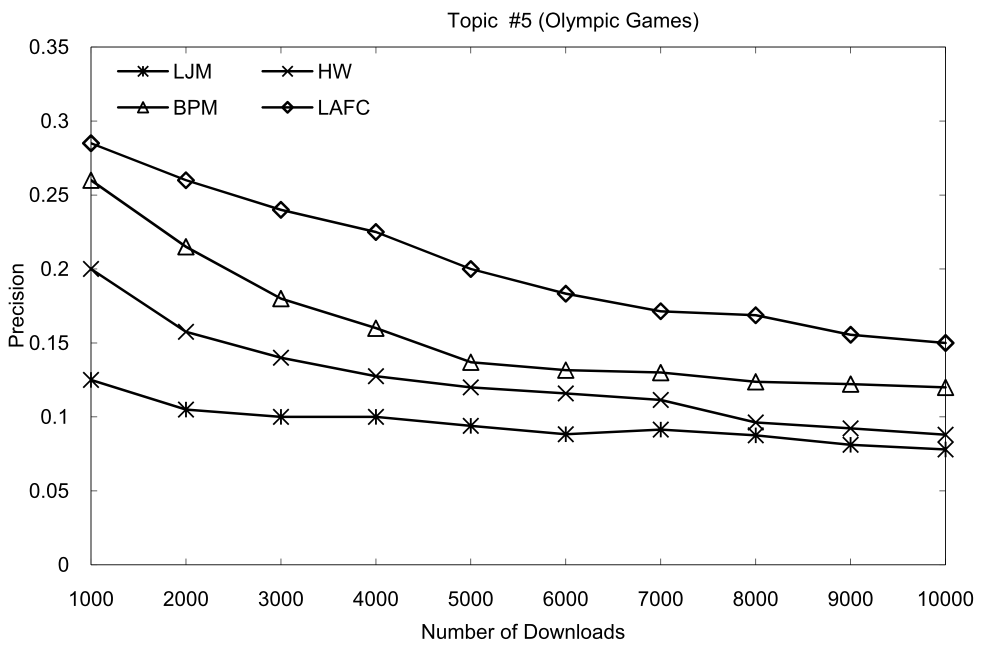

Fig. 6 Precision of crawlers for search topic #5 (Olympic Games)

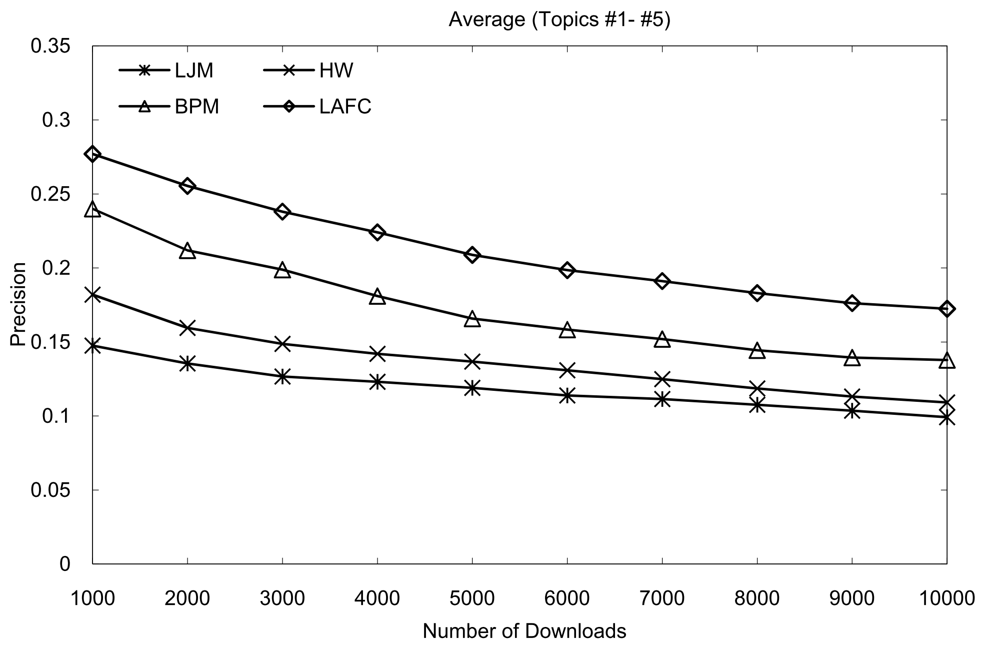

Fig. 7 Precision of crawlers averaged over all search topics

sults given in the third column (labeled as "Test results"), the ranking of each crawler is computed and summarized in the last column of Table 4. As expected, LAFC has the best result in terms of precision and is ranked at the highest position and LJM [8] has the worst results.

5.3 Experiment III

This set of simulation experiments is conducted to evaluate the performance of the proposed focused crawler in terms of recall. LAFC is compared with BPM [6], LJM [8], and HW [12] and the obtained results are summarized in Table 5. As mentioned earlier, the relevant set is defined as the set of all documents that are returned by different crawlers and indexed as "relevant". A document is indexed "relevant", if its similarity is not less than 0.75. In these experiments, learning rate is set to 0.09, control threshold is fixed at 0.8, and the number of downloads is 10,000 for each search topic. Comparing the results of different crawlers for different search topics, it can be seen that the proposed crawler significantly outperforms the others. This is expected from the significantly higher precision rate of the proposed crawler shown in Figs. 2–7 and Table 4. Numerical results show that the

Table 4 Statistical significance of the precision

Crawler x	Mean precision (x) $(\pm SD)$	Test results				Ranking
		Crawler y	Mean precision $(x-y)$	Difference significance	Performance	
BPM [6]	0.1380 (±0.0115)	LJM [8]	0.0388	3.45E−06	+	2
		HW [12]	0.0284	2.71E−02	+	
		LAFC	−0.0345	1.15E−06	−	
LJM [8]	0.0992 (±0.0142)	HW [12]	−0.0104	3.82E−01	−	4
		LAFC	−0.0732	6.00E−09	−	
HW [12]	0.1096 (±0.0333)	LAFC	−0.0628	1.91E−04	−	3
LAFC	0.1724 (±0.0053)					1

Table 5 Recall rate of different crawlers for search topics #1–#5

Crawler	Recall				
	Linux	Robotics	Java Programming	Asthma	Olympic Games
BPM [6]	0.73	0.63	0.62	0.61	0.65
LJM [8]	0.56	0.41	0.42	0.48	0.42
HW [12]	0.61	0.42	0.48	0.54	0.47
LAFC	0.85	0.78	0.76	0.83	0.81

proposed crawler is capable of finding more than 75 % of the relevant documents for all search topics. This increases to a good rate 85 % for search topic "Linux". This is because the proposed crawler learns how to construct the download paths including the most relevant documents. For all search topics, as expected Table 5 shows that BPM [6] is ranked below LAFC, HW [12] is ranked below BPM [6], and LJM [8] has the worst results.

5.4 Experiment IV

Any-time behavior is another property that is shown for different crawlers in Fig. 8 by plotting the development of the number of relevant downloaded documents in time. This property is useful to understand how fast each crawler finds the relevant documents. Figure 8 shows the number of relevant crawled documents, where the execution time changes from 100 (min) to 1900 (min) with increment step 200 and learning rate and control threshold are set to 0.09 and 0.8, respectively to make a good trade-off between the precision rate and the running time. Comparing the curves shown in Fig. 8, it can be seen that the proposed crawler has the shortest crawling time. That is, the number of relevant documents crawled by the proposed technique for a given time interval is significantly higher than those of the other crawlers. This is because the proposed crawling technique avoids visiting the off-topic documents by learning the download paths leading to the large set of relevant documents. This improves the crawling speed and precision of LAFC. Figure 8 implicitly shows the precision of the crawlers. The results given in this figure show that BPM [6] outperforms LJM [8] and HW [12] in terms of precision and LJM [8] has the worst results. No changes can be seen in the curve of LAFC for times later than 1100 (min). This curve shows a rapid progress in the beginning and flattening out later on (after 1100 min). This is because LAFC crawls all 10,000 documents before this time. Running time of all crawlers is given in Table 6 in more detail. The curves of the other crawlers in Fig. 8 show LJM [8] completes after LAFC and before HW [12], and BPM [6] has the longest running time.

Table 6 includes the detailed running time (in min.) of the proposed crawler for different search topics in comparison with BPM [6], LJM [8], and HW [12], where each crawler downloads 10,000 documents. The proposed crawler adaptively updates its configuration according to the Web dynamic so that it converges to the most promising paths leading to the large set of on-topic documents. Therefore, as the results shown in Fig. 8 confirm, the proposed crawler is capable of finding the most relevant documents in an as short as possible running time. From Table 6, it can be seen that for all search topics the running time of the proposed crawler is significantly shorter than that of the other crawlers. The obtained results also show that the simple crawling technique used in [8] outperforms both BPM [6] and HW [12]

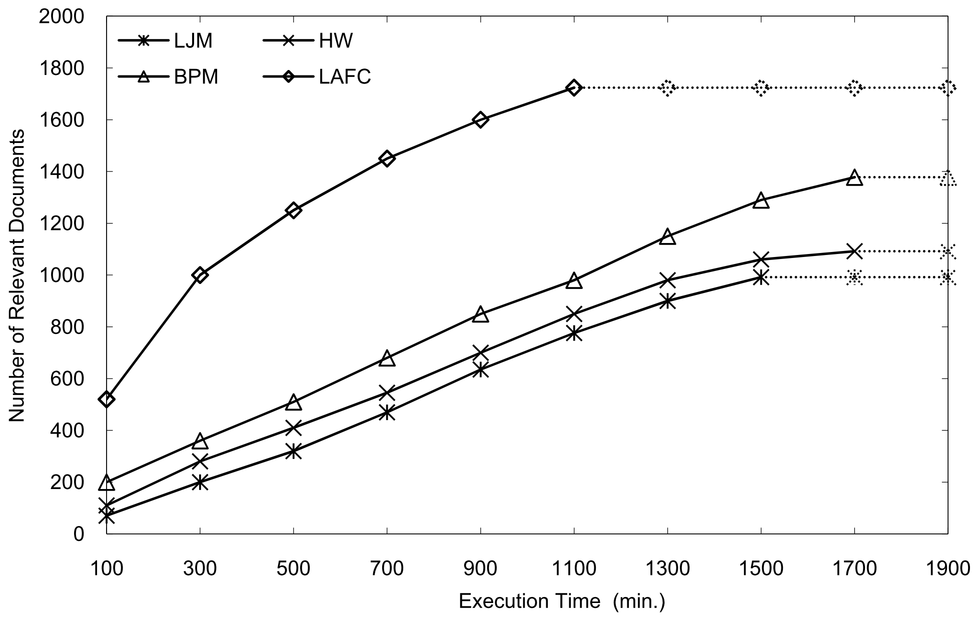

Fig. 8 The number of relevant downloaded documents versus execution time

Table 6 Running time of different crawlers for search topics #1–#5

Crawler	Running time (min)				
	Linux	Robotics	Java Programming	Asthma	Olympic Games
BPM [6]	1419.31	1693.11	1878.34	1616.53	1520.12
LJM [8]	1295.15	1390.28	1555.56	1391.98	1397.95
HW [12]	1382.42	1503.42	1690.01	1493.45	1432.22
LAFC	934.971	902.352	1180.67	1097.23	1009.15

in terms of running time (while it has the lowest precision rate), and BPM [6] which is a more detailed extension of LJM [8] is the most time consuming method.

6 Conclusion

In this paper, a decentralized algorithm was proposed for focused Web crawling based on learning automata theory. In the proposed learning crawler, each Web document is associated with an automaton. The crawler starts searching the topic from the seed URL and the learning automaton of each downloaded document randomly selects one of its URLs (hyperlinks) as its actions. The document corresponding to the chosen hyperlink is crawled and this process continues if the relevancy degree of the document to the search topic does not fall below a certain threshold. The selected URL is rewarded if the similarity between the search topic and document is larger than those seen so far. Otherwise it is penalized. By this, each learning automaton learns the most relevant URLs of its corresponding document and so the set of cooperating learning automata (NLA) learns the promising paths leading to the target on-topic documents. The proposed crawler easily and quickly updates its configuration and therefore it effectively adapts to the Web dynamic. The proposed crawler harvests the Web documents having a desired level of relevancy only. Therefore, it is expected to have a higher precision rate because of construction a small Web graph of only on-topic documents. It was shown that by the proper choice of the learning rate, the crawler selects the most topic-relevant URL of each Web document with a probability as close to one as possible. To show the performance of the proposed crawler a set of simulation experiments was conducted. The results of the proposed crawler were compared with those of BPM [6], LJM [8] and HW [12]. Simulation study showed that the proposed crawler significantly outperforms the other crawlers in terms of precision, recall, and running time. The t-test was used to verify the statistical significance of the precision results of the proposed crawler.

References

1. Akbari Torkestani J (2012) LAAP: a learning automata-based adaptive polling scheme for clustered wireless ad-hoc networks. Wirel Pers Commun (in press)

2. Akbari Torkestani J (2012) Mobility prediction in mobile wireless networks. J Netw Comput Appl (in press)
3. Jung JJ (2009) Using evolution strategy for cooperative focused crawling on semantic web. Neural Comput Appl 18:213–221
4. Rungsawang A, Angkawattanawit N (2005) Learnable topic-specific web crawler. J Netw Comput Appl 28:97–114
5. Batzios A, Dimou C, Symeonidis AL, Mitkas PA (2008) BioCrawler: an intelligent crawler for the semantic web. Expert Syst Appl 35:524–530
6. Batsakis S, Petrakis EGM, Milios E (2009) Improving the performance of focused Web crawlers. Data Knowl Eng 68:1001–1013
7. Zhang H, Lu J (2010) SCTWC: an online semi-supervised clustering approach to topical web crawlers. Appl Soft Comput 10:490–495
8. Liu H, Janssen J, Milios E (2006) Using HMM to learn user browsing patterns for focused Web crawling. Data Knowl Eng 59:270–291
9. Akbari Torkestani J, Meybodi MR (2012) Finding minimum weight connected dominating set in stochastic graph based on learning automata. Inf Sci (in press)
10. Akbari Torkestani J (2012) An adaptive learning automata-based ranking function discovery algorithm. J Intell Inf Syst (in press)
11. Patel A, Schmidt N (2011) Application of structured document parsing to focused web crawling. Comput Stand Interfaces 33:325–331
12. Hsu C-C, Wub F (2006) Topic-specific crawling on the web with the measurements of the relevancy context graph. Inf Sci 31:232–246
13. Akbari Torkestani J (2012) A stable virtual backbone for wireless MANETS. Telecommun Syst J (in press)
14. Menczer F, Pant G, Srinivasan P (2004) Topical web crawlers: evaluating adaptive algorithms. ACM Trans Internet Technol 4(4):378–419
15. Ehrig M, Maedche A (2003) Ontology-focused crawling of web documents. In: Proceedings of the symposium on applied computing (SAC 2003)
16. Hliaoutakis A, Varelas G, Voutsakis E, Petrakis EGM, Milios E (2006) Information retrieval by semantic similarity. Int J Semantic Web Inf Syst 3(3):55–73
17. Pant G, Srinivasan P (2005) Learning to crawl: comparing classification schemes. ACM Trans Inf Syst 23(4):430–462
18. Diligenti M, Coetzee F, Lawrence S, Giles C, Gori M (2000) Focused crawling using context graphs. In: Proceedings of the 26th international conference on very large databases (VLDB 2000), pp 527–534
19. Symeonidis AL, Valtos V, Seroglou S, Mitkas PA (2005) Biotope: an integrated simulation tool for augmenting the intelligence of multi-agent communities residing in hostile environments. IEEE Trans Syst Man Cybern, Part A 35(3):420–432. Special Issue on Self-organization in Distributed Systems Engineering
20. Salton G, Wong A, Yang CS (1975) A vector space model for automatic indexing. Commun ACM 18(11):613–620
21. http://dir.yahoo.com
22. Narendra KS, Thathachar MAL (1989) Learning automata: an introduction. Prentice-Hall, New York
23. Lakshmivarahan S, Thathachar MAL (1976) Bounds on the convergence probabilities of learning automata. IEEE Trans Syst Man Cybern SMC-6:756–763
24. Billard EA, Lakshmivarahan S (1999) Learning in multi-level games with incomplete information—Part I. IEEE Trans Syst Man Cybern, Part B, Cybern 19:329–339
25. Akbari Torkestani J, Meybodi MR (2010) A new vertex coloring algorithm based on variable action-set learning automata. J Comput Inform 29(3):447–466
26. Akbari Torkestani J, Meybodi MR (2010) An efficient cluster-based CDMA/TDMA scheme for wireless mobile AD-hoc networks: a learning automata approach. J Netw Comput Appl 33:477–490
27. Meybodi MR (1983) Learning automata and its application to priority assignment in a queuing system with unknown characteristics. PhD thesis, Department of Electrical Engineering and Computer Science, University of Oklahoma, Norman, Oklahoma, USA
28. Hashim AA, Amir S, Mars P (1986) Application of learning automata to data compression. In: Narendra KS (ed) Adaptive and learning systems. Plenum Press, New York, pp 229–234
29. Oommen BJ, Hansen ER (1987) List organizing strategies using stochastic move-to-front and stochastic move-to-rear operations. SIAM J Comput 16:705–716
30. Unsal C, Kachroo P, Bay JS (1999) Multiple stochastic learning automata for vehicle path control in an automated highway system. IEEE Trans Syst Man Cybern, Part A 29:120–128
31. Barto AG, Anandan P (1985) Pattern-recognizing stochastic learning automata. IEEE Trans Syst Man Cybern SMC-15:360–375
32. Thathachar MAL, Harita BR (1987) Learning automata with changing number of actions. IEEE Trans Syst Man Cybern SMG17:1095–1100
33. Xiao L, Wissmann D, Brown M, Jablonski S (2004) Information extraction from the web: system and techniques. Appl Intell 21(2):195–224
34. Camacho D, Aler R, Borrajo D, Molina JM (2006) Multi-agent plan based information gathering. Appl Intell 25(1):59–71
35. Santos E, Santos EE, Nguyen H, Pan L, Korah J (2011) A large-scale distributed framework for information retrieval in large dynamic search spaces. Appl Intell 35(3):375–398
36. Kim S, Zhang BT (2003) Genetic mining of HTML structures for effective Web-document retrieval. Appl Intell 18(3):243–256
37. Akbari Torkestani J (2012) Degree constrained minimum spanning tree problem in stochastic graph. J Cybern Syst 43(1):1–21
38. Akbari Torkestani J, Meybodi MR (2011) LLACA: an adaptive localized clustering algorithm for wireless ad hoc networks based on learning automata. J Comput Electr Eng 37:461–474
39. Akbari Torkestani J, Meybodi MR (2011) A link stability-based multicast routing protocol for wireless mobile ad hoc networks. J Netw Comput Appl 34(4):1429–1440
40. Akbari Torkestani J (2012) An adaptive backbone formation algorithm for wireless sensor networks. Comput Commun (in press)
41. Akbari Torkestani J (2012) A new approach to the job scheduling problem in computational grids. J Clust Comput (in press)