Journal of Web Engineering, Vol. 17, No.3&4 (2018) 270-283

HIDDEN WEBPAGES DETECTION USING DISTRIBUTED LEARNING AUTOMATA

MANISH KUMAR
PEC University of technology, Chandigarh India
manishkamboj3@gmail.com

RAJESH BHATIA
PEC University of technology, Chandigarh India
rbhatiapatiala@gmail.com

Received July 7, 2017
Revised January 22, 2018

Webpages directly connected to each other on the Web can be reached easily by following hyperlinks. Those webpages that are not linked by hyperlinks comprises hidden Web and it is challenging to find them. Furthermore, most of webpages in hidden Web are generated dynamically. This paper proposes first time an algorithm to find webpages in hidden Web using distributed learning automata. Learning automata use its self-learning characteristic of taking action based on the action probabilities using <keyword-value> pairs. These actions may lead the current webpage to hidden webpages that are generated dynamically. At each stage of the proposed algorithm, we determine the edge that should be chosen to reach webpage of interest. The proposed algorithm is validated on four different websites from dmoz.org. Precision-recall curve and coverage plot in the results section shows the effectiveness of the proposed algorithm.

Keywords: hidden Web; learning automata; distributed learning automata; DLA
Communicated by: B. White & E. Mendes

1 Introduction

It is easy to locate any information on World Wide Web (WWW) using search engines. When a user seeks some information, only Publically Indexable Web (PIW) is searched. While the PIW has trillion of linked HTML webpages, tonnes of information is "hidden" in the hidden or deep Web. It also includes webpages or information that is behind query forms or login forms. However, hidden Web mainly comprises of webpages that are generated dynamically [1]. Our proposed algorithm will uncover the dynamically generated webpages. Hidden Web is important, as the information in it is around 500 times larger than PIW. Using overlap analysis, a white paper [2] estimated the size of dynamically generated webpages to be around 7,500 terabytes. Also, the information in the hidden Web is growing exponentially. This large size of information in hidden Web raises an urgent need to have an algorithm for acquiring dynamically generated webpages.

As already stated the part of Web that past login form, search and query interfaces is called hidden Web. This part of Web is not accessible by following hyperlinks present on the webpages. Traditional

crawler moves from one webpage to other by using these hyperlinks only and hence they cannot reach to hidden Web [1]. The content of the hidden Web can be accessed by using a direct URL or these can be generated by some user action. Crawling hidden Web is a challenging task because: firstly, the scale of information on hidden Web is very large and secondly, on some website there are restrictions on accessing data of hidden Web.

This paper uses Learning Automata (LA) to find webpages in hidden Web. The problem this paper handles is "to find webpages from hidden Web that are generated dynamically by performing an action on the current webpage." To solve this issue, LA is used as it can model dynamically generated webpages in hidden Web effectively. LA has a finite set of states, transitions between various states are using the transition map. These webpages are viewed as the states of LA that can be reached from the starting state by using a set of actions. The states of LA represent two types of webpages: PI and hidden webpages. Hidden webpages are considered as those states of LA that can only be reached from a current state by making a transition based on some action probability. The action probabilities of the LA are updated on reaching to a new webpage that is represented as another state of LA. Section 3 discusses the detailed algorithm for exploration of dynamic webpages. The webpage reached may be PI webpage or hidden webpage depending upon the type of link used to reach upto that webpage.

In the literature, learning automata have been successfully utilized in some applications like solving the NP-Complete problem [3], network routing [4], capacity assignment [5] and neural network [6], [7]. Learning automata is also used for solving uniform partitioning method [3]. Given a graph, the motive is to partition the nodes into two equal size sets to minimize the sum of cost of edges having endpoints in different sets. An adaptive routing in telephone network using learning methods is used in [4]. It uses an algorithm that updates the routing probabilities on the basis of network feedback. Capacity assignment [5] deals with the problem in a prioritized network. It focuses on finding the links with a best possible set of capacities that satisfies the traffic requirement while minimizing the cost. They propose three different solutions to this problem. Meybodi and Beigy [6] discussed a new learning automata based algorithm for adaptation of backpropagation parameters. The various parameters that are used include learning rate, momentum factor and steepness. Authors also study the ability of learning automata based schemes in escaping from local minima when backpropagation fails to find the global minima [7]. The remainder of the paper is organized as section 2 discusses the basis of learning automata and its various tuples, section 3 presents the proposed hidden Web distributed learning automata and algorithm for finding the hidden webpages. Result and analysis of the proposed algorithm are given in section 4.

2 Learning Automata

LA is an abstract model of a finite state machine that can perform a particular action from finite set of actions. These actions are chosen either randomly or according to a fixed initial probability. For the action chosen, the neighboring LAs that constitute the environment generate response as reward or penalty. The feedback of environment is used to select next action to be taken by LA. During this process, LA learns to choose the best action from an allowed set of actions [8]. Figure 1 depicts the interaction between LA and its environment. LA chooses an action from a finite set of actions with some probability and passes it to the environment. The environment consists of neighboring LAs that in turn generate a response that is given back as feedback to the action generating LA.

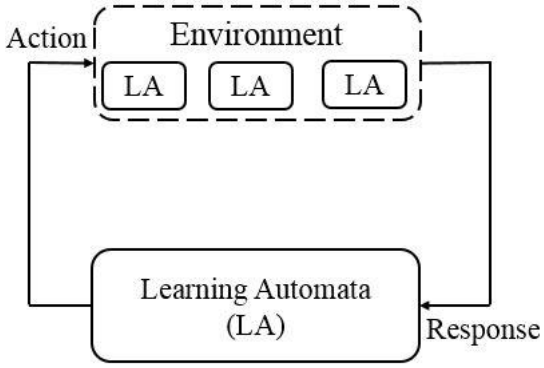

Figure 1 Interaction between LA and its environment.

A fixed structure stochastic learning automata can be defined by using five tuples (α, Φ, β, F, G) where

i. $\alpha = \{\alpha_1, \alpha_2 \ldots \alpha_r\}$ is the finite set of actions that can be chosen with associated action probability vector p.

ii. $\Phi = \{\Phi_1, \Phi_2 \ldots \Phi_3\}$ is the set of states of the automata.

iii. β is the set of inputs that can have two values 0 for penalty and 1 for reward.

iv. F is the transition map. It maps the current state to the next state on receiving an input. Mathematically, $F: \Phi \times \beta \rightarrow \Phi$.

v. G is the output map that determines the action taken by the automata if it is in some state. Mathematically, $\Phi \rightarrow \alpha$.

As it is evident from the definition abovementioned automata is deterministic. The action is input $\alpha(k)$ at a given time k to environment that gives back response $\beta(n)$ to the automata. Depending upon the response generated, environment penalizes or rewards the automata according to equations 1 and 2. At time $(k+1)$ state of the automata, become $\Phi(k+1)$ according to a new action chosen. Also, the penalties and rewards values are chosen randomly at initial point and it is desired that the interaction with the environment always rewards the automata.

The automata action probability vector is always updated depending upon the β or output from the environment. Learning algorithm of the automata is a recurrence relation and is used to modify the action probability vector p as given below [9]. Let $\alpha_i(k)$ is the action chosen by LA at time k then

$$P_j(k+1) = \begin{cases} P_j(k) + a \times \left[1 - P_j(k)\right] & if\ i = j \\ P_j(k) - a \times P_j\ (k) & if\ i\ \neq j \end{cases} \tag{1}$$

When β(k) = 0 i.e. penalty.

$$P_j(k+1) = \begin{cases} P_j(k) \times (1-b) & if\ i = j \\ \dfrac{b}{r-1} + P_j(k)(1-b) & if\ i\ \neq j \end{cases} \tag{2}$$

When β(k) = 1 i.e. reward.

Parameters $0 < b \ll a < 1$ where a and b are the regularization terms for determining the increase or decrease in the action probabilities. The values of a and b can be decided based on learning rate in terms of iterations of algorithm, its accuracy and number of nodes under consideration. Also, r is the number of actions for LA from set α. Further if $a == b$, this algorithm is called linear reward penalty and if $b == 0$ it is called linear reward inaction. The action selected by the neighboring LAs of any particular LA determines the signal to the LA and constitutes its environment.

2.1 Distributed Learning Automata

A basic LA can be considered as a simple agent that can do simple things. The full potential of the LA is recognized when they are interconnected to work together. A Distributed Learning Automaton (DLA) is a network of LAs cooperating to solve a problem. In such a distributed network, each time just one automaton is active. The number of actions that may be performed by an automaton is equal to the LAs connected to it. The model of DLA network is shown in figure 2, it represents a graph in which each vertex is an automaton.

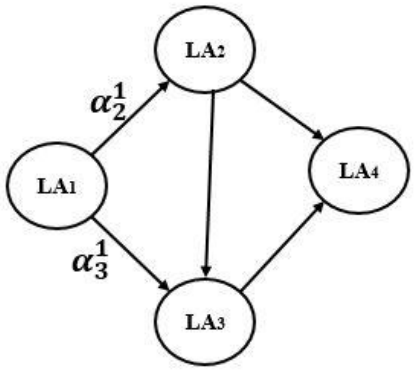

Figure 2 A DLA with four learning automata.

As shown in figure 2, nodes represent LA_i i.e. a learning automaton and an edge between nodes LA_i and LA_j shows the action α_j^i in LA_i that triggers the node LA_j. The action probability vector will be $p^k = \{p_1^k, p_2^k \ldots p_r^k\}$ such that $p_m^k$is the probability of action α_m^k that triggers LA_m [10]. Number of actions for an LA is equal to number of other LAs connected to it. Any action chosen by an LA activates a connected LA on the other side of edge. The working of DLA graph can be explained as: In the starting, one of the LA chooses an outgoing action edge according to the action probability. The LA on the other side of action edge gets activated and is added to the list of explored nodes. This process of choosing an action edge and activating another LA is repeated until all the nodes in the DLA graph are explored. So, starting from one of the webpage (considered as node) other webpages (nodes) are explored using action probabilities. In hidden Web, when a user is on a particular webpage other webpages can be generated by performing some action. This action can be a form-filling event, value submission, query form submission or any event that generate a new hidden webpage. The edge between webpages does not necessarily represent a hyperlink between them. Edge between a PI webpage and hidden webpage represent the action edge, the action that needs to be taken by the user to reach to that particular hidden webpage. A user can pass some values to the current webpage that result in the generation of a new webpage. The action is considered as the set of values passed by the user for the generation of webpage. The next section discusses the detailed algorithm for the exploration of dynamically generated webpages in hidden Web.

3 The Proposed Algorithm

In this Section, we propose an algorithm to find webpages from hidden Web using distributed learning automata. The input to our proposed automata named Hidden Web Distributed Learning Automata (HWDLA) will be a starting URL or seed URL of the website from which hidden webpages are to be uncovered. It is to be noted that webpages in any website imprint a DLA graph structure [11]. Any website, considered as DLA has two types of webpages assumed as nodes: PI webpages that are connected by hyperlinks and hidden webpages that can be reached only when any of the PI nodes perform some action. Further, the DLA is assumed to have two types of edges: first is a boldface edge that connects any two PI webpages. Second is a washy edge that connects PI webpages to the hidden webpages as shown in figure 7. These two type of edges are adopted to distinguish hyperlink and action edge between the webpages. The purpose of our algorithm is to find hidden webpages from any given website assuming DLA structure. We believe this is the first attempt to visualize the hidden Web from this point of view.

The proposed HWDLA can be defined by using 7 tuples (Φ, Υ, £, α, β, Q, G):

i. Φ is the set of states where each state represents a webpage in the given website. This includes both PI webpages and hidden webpages.

ii. Υ: $\Upsilon \subset \Phi$ is the set of hidden states. The webpages in this set are hidden and are not accessible by following the hyperlinks. This set is assumed to be unknown at the beginning and purpose of the proposed algorithm is to uncover this set.

iii. £: £ $\in \Phi$ is the starting state or the seed URL of the website under consideration. It is to be noted that £ $\notin \Upsilon$ as seed URL can never be from hidden states.

iv. $\alpha = \{\alpha_1, \alpha_2 \ldots \alpha_r\}$ is the finite set of actions that can be chosen with associated action probability vector p for each webpage. In this case, the action will be <keyword-value> pair that are required to generate the hidden webpage. These pairs are stored in a table called Labelled Value Set (LVS) which have keywords and their corresponding values that need to be passed on to the current webpage for generating a hidden webpage. The LVS table have labelled keywords and corresponding values that can be used to fill the forms found on PI webpages to reach hidden webpages. LVS table is used only when the domain of the form element is infinite, it is not required in case of finite domain elements like dropdown list, selection list, radio button etc.

v. β is the set of inputs that can have two values 0 for penalty and 1 for reward.

vi. Q: $(\Phi - \Upsilon) \times \alpha \rightarrow \Phi$ is the transition function that explains how the next webpage (hidden webpage desired) is generated from the current webpage. It is to be noted that the range of this function Φ specifies that it is always not necessary that from the current webpage, hidden webpage will be reached. This function will be explained in detail, later in this section.

vii. G is a function that decides the set of states $\{\Phi_{(j-1)M+1} \ldots \Phi_{jM}\}$ for each action α_j such that M is output state of the initial state.

$$G(\Phi_i) = \alpha_j \qquad \text{if } (j-1)M + 1 \leq i \leq jM \tag{3}$$

This G can be adjusted so that from PI states we always reach to hidden states. LA in DLA will choose α_1 if it is in any of the first M states. α_2 will be chosen if it is in any of the states from $\Phi_{M+1}$to Φ_{2M}. In this way for every action, we have fixed action that can be performed in association with some particular states from where we are starting.

When starting from a given seed URL or state £ the starting state can make any transition depending upon the action chosen. HWDLA learns that transitions from any state in Φ should result in the exploration of a state from ϒ using G. Also, G function decides the action with some probability and LA is either penalized or rewarded as in equation 4, 5 and 6 by modifying equation 1 and 2 [12]. If we move from webpage σ_k to σ_m then LA_k updates the probability vectors p_m^k of action α_m^k as follows:

$$p_m^k(n+1) = p_m^k(n) + a_m^k[1 - p_m^k(n)] \tag{4}$$

$$p_{ij}^k(n+1) = (1 - a_m^k)p_j^k(n) \;\; j \neq m \quad \forall j \tag{5}$$

$$a_m^k = \frac{E_m^k}{1+E_m^k} \tag{6}$$

$$E_m^k = -(p_m^k \log p_m^k + (1 - p_m^k)\log(1 - p_m^k)) \tag{7}$$

The value of E_m^k in equation 7 denotes the relation between webpage σ_k to σ_m. In equation 6 the regularization term also increases as the probability among webpages is updated. This decision of penalizing or rewarding a LA is taken based on the action edge being explored is boldface or washy. The algorithm rewards the LA on exploring a node connected by washy edge and penalizes it on exploring a node connected by boldface edge.

Now, function Q for each operation is explained in detail.

i. ***Transitions for rewards: webpages from the hidden Web found without error by PI webpage***: This is the case when making a transition from a webpage in the set Φ to any state $\Phi_{(j-1)M+1}$ results in a webpage from ϒ. In this case, from a PI webpage we have found a hidden webpage.

ii. ***Transitions for rewards: webpages from the hidden Web found without error by another hidden webpage***: This is the case when a webpage Υ_j is generating another webpage Υ_{j+1} for an action Φ_j.

iii. ***Transitions for penalties: PI webpage transit to another PI webpage***: As we are only interested in finding the hidden webpages. In this case, a PI webpage makes a transition to another PI webpage instead of making transition to a hidden webpage. This is the case encountered when Φ_i is the state such that $\Phi_i \in \{ \Phi_{(k-1)M+1} \dots \Phi_{kM}\}$ and makes a transition to $\Phi_j \in \{ \Phi_{(j-1)M+1} \dots \Phi_{jM}\}$.

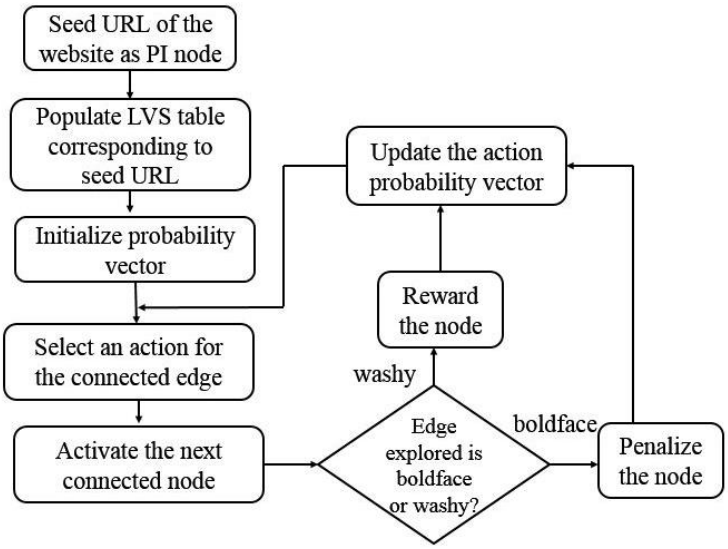

Figure 3 Flowchart for the proposed algorithm.

Depending upon the response generated the probability vector is updated so that the next correct action can be taken. The flowchart for the proposed algorithm is shown in figure 3. As shown, after the initialization, a connected edge is activated according to action probability. Decision block corresponds to the penalty or reward accomplishment depending upon the edge explored is washy or boldface. The more detailed description of the algorithm is given next.

The algorithm 1 can be described as follows: At first, a network of LA is created based on the webpages of the website whose seed URL is provided. In DLA, each node represents an LA and each outgoing edge from the node is one of the actions that can be taken by LA. At time k, source LA that is the starting node £ of the graph chooses one of its action by using the action probability vector and using the values in the LVS, say action α_m. This action activates the other LA Φ_k on the other side of edge starting from £. Depending upon whether this edge is boldface or washy edge, the environment penalizes or rewards the automaton. This corresponds to the webpage explored is PI or hidden webpage. The process of choosing an action and activating an LA is repeated until all the LAs on the other side of washy edges are reached or a threshold number of iterations are reached.

Algorithm 1: Algorithm for finding webpages in the hidden Web

Input: A seed URL $£ \in \Phi$ and $\alpha = \{\alpha_1, \alpha_2 \ldots \alpha_r\}$ with LVS having preliminary values.

Output: The set τ of webpages explored from the hidden Web $\Upsilon \supseteq \tau$.

Construct a DLA for the current set of webpages with given seed URL taking $£$ as the starting state.

Let k be the step number, initialized to 0.

Initialize the probability vector.

repeat

Taking $£$ as the source node

while ($\tau \neq \Upsilon$ or $k \leq k_{threshold}$)

do

For the present state Φ_j, selects an action from α_j^i where i varies upto connected neighbors of Φ_j.

Using function G determines the next feasible state Φ_s.

if (Φ_s is connected to Φ_j with washy edge)

Reward the selected action as case (i) and (ii) of Q

Update the probability vector and action values

if ($\Phi_s \notin \tau$)

Add Φ_s to τ

else

Penalize the selected action as case (iii) of Q

Update the probability vector and action values

end if

Increment the step number k

end do while

When some of the parameters are unknown and stabilizing input can be generated by following the estimation of unknown parameters based on the repeated trials LA is the best option to use [8]. The main objective of the proposed algorithm is to determine how the past actions and responses guide the current

action and state. To compare various DLA we consider $\boldsymbol{M}(\boldsymbol{n})$ as the average penalty for a given action probability vector.

$$M(n) = E[\,\beta(n)/p(n)] = Pr[\,\beta(n) = 1/p(n)] \tag{8}$$

$$= \sum_{i=1}^{r} \Pr[\,\beta(n) = 1/\alpha(n) = \alpha_i]\,\Pr[\alpha(n) = \alpha_i] \tag{9}$$

where β, α are the input set and action set respectively. Further, as a website can have vast number of webpages, the LA becomes ϵ-optimal i.e. $\lim_{n\to\infty} E[M(n)] < c_{l+\epsilon}$, where $c_1, c_2 \ldots c_r$ are the penalty probabilities and ϵ is any arbitrary value $\epsilon > 0$ chosen by a proper choice of automaton parameters. Performance of any LA is mainly affected by two factors, first is the initial condition and second is the set of penalty probabilities of the environment. The above algorithm was implemented and tested to check its applicability to explore hidden webpages.

3.1 Learning methodology

In the first step, a learning automaton is assigned to each vertex of the graph and action probability is initialized. The following steps are repeated until the stopping criteria are met. The stopping criteria are predefined as a total number of iterations or when the value of probability vector exceeds a threshold.

Various LAs are adapted to learn through reinforcement learning. Here, the meaning of reinforcement learning is making an optimal decision using rewards or punishment received from neighboring LAs. Unlike supervised learning in this learners are never told about the correct action to a particular state. The LA uses feedback from the environment to learn about the action expressed in terms of scalar reward.

Learner uses a set of PI states which are webpages already retrieved i.e. $w \in (\Phi - \Upsilon)$ and a set of hidden webpages Υ. The goal of reinforcement learning is to learn a policy Ω that maps $(\Phi - \Upsilon) \to \Upsilon$ and maximizes the sum of its rewards over time. Value of long term reward is computed by summing up the rewards from each step.

An infinite-horizon discounted model is used for computing the overall value of reward. Consider a starting state s, using policy Ω we will get a sequence of state-action pairs (x_t, y_t). The sequence of reward is given by

$$R_t = r_t(x_t, y_t)\ where\ t = 1, 2, 3 \ldots\ldots$$

The total reward from the policy Ω will be

$V_\lambda^\Omega(s) = \lim_{N\to\infty} E_s^\Omega\,[\sum_{t=1}^{N} \lambda^{t-1}\, r(x_t, y_t)]$ where $\lambda \in [0,1)$ is the discount factor.

An evaluation is said to be optimal if it maximizes the overall reward. If $Q^*(s, l)$ denotes value of choosing an action l from a starting state s and afterward it follows the optimal policy. So the overall value become $Q^*(s, l) = R(s, l) - \lambda V^* T(s, l)$ where R is the reward function and T is any other transition. Here $V^*$is the total value from each node to other node $V^\Omega(s) = \sum_{t=0}^{N} \lambda r_t$. Where r_t is the reward action-state pair after t time stamps and following policy Ω. So Q function maps an action to a scalar value.

4 Results and Discussions

For validating the effectiveness of the proposed algorithm, it is implemented and various outcomes are discussed in this section. All experiment were performed on Intel Xenon CPU E5620 having 20 GB of RAM running Windows Server 2012 R2 standard. For our experiment, we took four random websites from following domains: Artificial Intelligence (Db1), Software (Db2), Home Automation (Db3) and Virtual Reality (Db4). The process of creation of a Database (Db) for these four websites was following: these domains were chosen randomly from dmoz.org (the site is officially closed from March 2017). From each randomly chosen domain, we extracted one website. We used in-house developed hidden Web crawler to crawl all type of webpages from these four websites. The webpages crawled includes both hidden and PI webpages. The crawler was able to detect webpages having forms of any type: Search form, login form, subscription form, single and multi-attribute forms.

These forms act as the entry point to the hidden Web. Further depending upon the domain of the website to be crawled, the crawler LVS table was populated to fill the values in the corresponding fields of the form. Also, all the websites were analyzed manually to include any webpage (hidden or PI webpage) that may be missed out by our crawler. Altogether, we collected a total of 58,251 webpages of which 29,395 were hidden webpages. This includes manually extracted webpages.

After gathering all the webpages, a graph structure was created in which hidden and PI webpages were marked separately, along with the edges connecting them. This was done using matplotlib in Python. A separate graph was created for each website. The initial state of each graph was the seed URL from where crawling was initiated. These graphs are used to validate our proposed algorithm for the exploration of hidden webpages from a website.

We used two standard metrics from the field of information retrieval for the evaluation of our proposed crawler: Precision and Recall. Precision with a threshold value p can be defined as the number of hidden webpages extracted correctly divided by total number of webpages extracted at or above threshold value p. Recall with a threshold value r can be defined as number of hidden webpages extracted correctly at or above threshold r divided by total number of webpages extracted at all probabilities. Figure 4 shows the Precision-Recall (PR) curve constructed by plotting precision-recall pairs that are obtained by using different threshold probabilities. PR curve illustrates the tradeoff between the two values as we vary the strictness of the algorithm.

To compute the value of precision and recall, we compared the number of hidden webpages extracted by our algorithm with the previously constructed databases. This is done for all the databases to compute precision recall value with a different threshold.

In figure 4, each point on the curve for a particular database shows precision and recall of finding a hidden webpage with a particular threshold probability. For example, the curve for Db1 has precision 0.86 at recall 0.1 and drops to 0.66, 0.65 at recall values of 0.2 and 0.3 respectively. For all the four

databases precision starts out high for low values of recall but decreases gradually as recall increases till a certain point. Precision value starts increasing again until it reaches a maximum and again decline a bit until the end. It is also to be noted that the precision value hits the lowest value of 0.62 for Db1. The performance of the proposed algorithm over four datasets is rather even. The precision is high in the starting as at initial time overall a few webpages are extracted. As the time passes and more webpages are extracted, value of recall increases and precision drops gradually. But after around halfway of recall the value of precision starts increasing again.

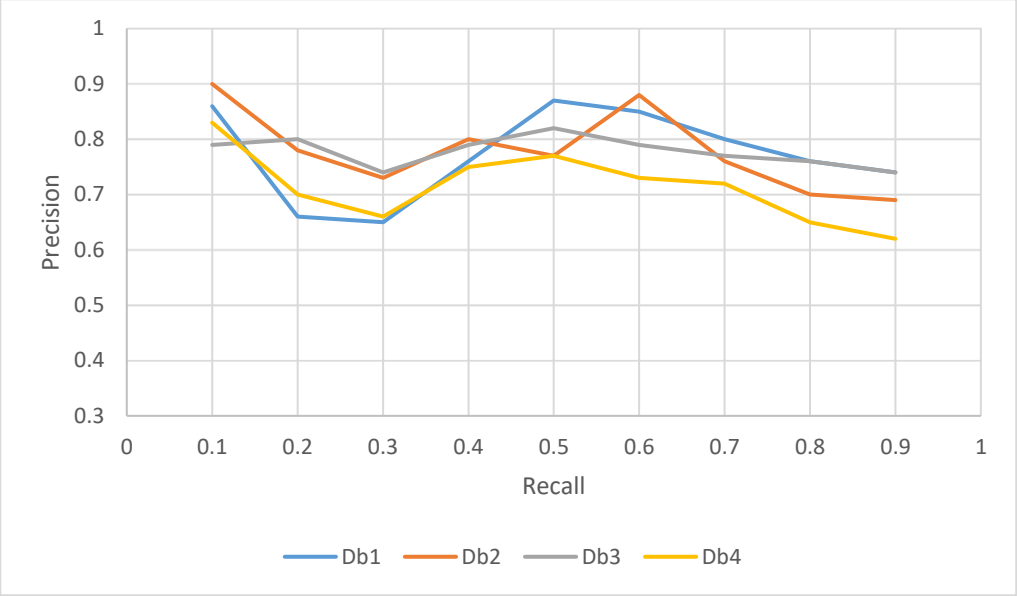

Figure 4 Precision-Recall curve for all the four databases.

In figure 5, we present the metric coverage for each database, coverage denotes the fraction of hidden webpages that can be found starting from seed URLs. In figure 5, the coverage is treated as a function of PI webpages. X-axis, represents a total number of PI webpages explored by the algorithm, and y-axis has the total number of hidden webpages explored. First observation from the graph is that for all databases hidden webpages are found after around 10% of PI webpages are explored, for Db3 it is around 30%. One of the reasons for such trend is that the seed URL always represent a PI webpage and further linked to many other PI webpages. It can also be concluded from figure 5 that with time more hidden webpages are explored but the number of hidden webpages always remain lower than the PI webpages. It is worth noticing that the overall we have around an equal number of PI and hidden webpages in the databases. The proposed algorithm explores an average of 90% of the hidden webpages.

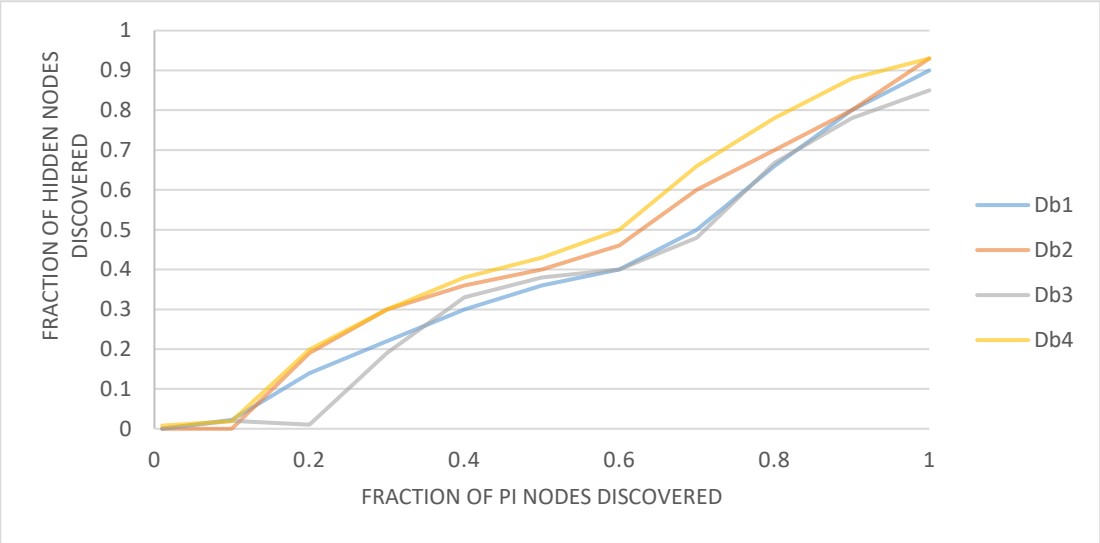

Figure 5 Hidden vs. PI nodes discovered by the algorithm.

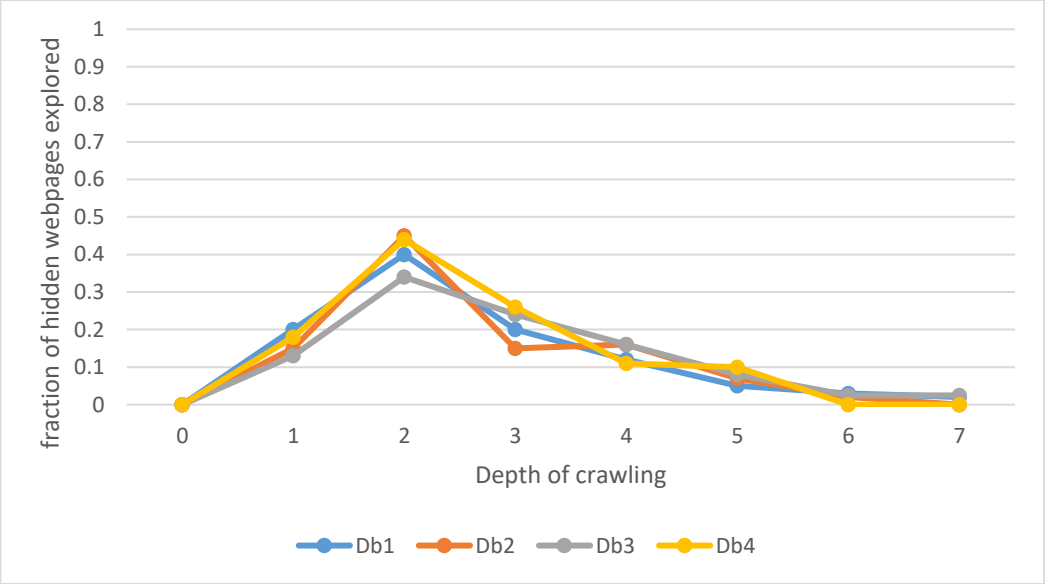

Figure 6 Depth of crawling vs. Hidden webpage explored.

Figure 6 reveals an interesting observation, considering 0 level as the starting of crawling i.e. seed URL, most of the hidden webpages are found at depth 2, also figure 6 represents that this behavior is repetitive for all 4 databases.

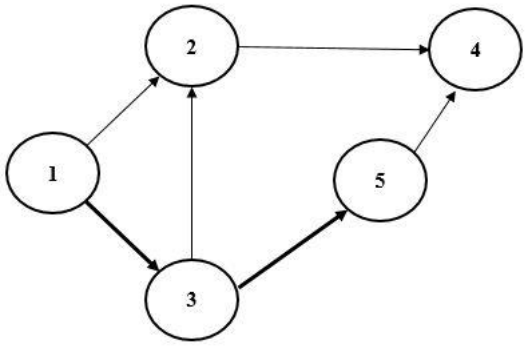

Figure 7 An example HWDLA.

To understand our algorithm in a better way, we use figure 7 for a dry run of the algorithm 1. Consider figure 7, in a stage say t, LA_1chooses one of its action α_2^1 that activates LA_2 connected by a washy edge. In the process of performing action the values from LVS table are also passed-on to reach LA_2. These values are dependent on the environment of LA_1. The process of selecting an action and activating the resultant LA is repeated until all the webpages of hidden Web namely 2 and 4 in this case, are found. Once we reach to LA_2 from $LA_1$the action probability is updated and LA_1 is rewarded.

In another case, it is entirely possible to choose action α_3^1 by LA_1 that activates LA_3 but as in this case the connected edge (LA_1, LA_3) is boldface. This action will penalizes the LA_1 and action probability will be updated. After reaching LA_3, there is a probability of choosing action α_2^3 that activates LA_2 that is already not being explored. Use of LA has not been explored earlier in hidden webpage exploration. So, we do not have sufficient work available in literature with which comparative analysis can be carried out.

5 Conclusions

The large size of important data present in hidden Web makes it necessary to crawl and index that data. The webpages in hidden Web are not connected by hyperlinks so it becomes difficult to get them. In this paper, a new algorithm for hidden webpages detection was proposed using distributed learning automata. Various tuples of proposed HWDLA were discussed alongwith the transitions for rewards and penalties. LAs were penalized or rewarded depending upon whether they reach to other LAs using boldface or washy edges. The main objective of the proposed algorithm is to determine how the past actions and response guide the current action and state. Various functions used by the algorithm are also explained. The performance of the proposed algorithm was compared using four different websites from dmoz.org. The metrics used for comparison include PR plot, coverage and depth at which hidden webpages were found. Our experimental results show that our strategy is effective in detecting hidden webpages. The proposed algorithm explores an average of around 90% of the hidden webpages.

Acknowledgement

We are thankful to the Department of Science and Technology, Government of India for financially supporting the research work presented in this paper.

References

[1] He B., Patel M., Zhang Z., and Chang K. C., Accessing the Deep Web : A Survey, 2000.

[2] Bergman M. K., White Paper: The Deep Web: Surfacing Hidden Value, J. Electron. Publ., 7(1), Aug. 2001.

[3] Oommen B. J. and de St. Croix E. V , Graph partitioning using learning automata, IEEE Trans. Comput., 45(2), Feb. 1996, 195–208.

[4] Srikantakumar P. R. and Narendra K. S., A Learning Model for Routing in Telephone Networks, SIAM J. Control Optim., 20 (1), 1982, 34–57.

[5] Oommen B. J., T Roberts. D., Continous learning automata Solutions to the capacity assignment problem, IEEE Transactions on Computers, 49(6), 2000, 608-620.

[6] Meybodi M. R. and Beigy H., New learning automata based algoritbms for adaptation of backpropagation algorithm pararmeters, Int. J. Neural Syst., 12(3), 2002, 45–67.

[7] Meybodi M. R. and Beigy H., A note on learning automata-based schemes for adaptation of BP parameters, Neurocomputing, vol. 48, 2002, 957–974.

[8] Narendra K. S. and Thathachar M. A. L., Learning Automata - A Survey, IEEE Trans. Syst. Man. Cybern., vol. SMC-4 (4), Jul. 1974, 323–334.

[9] Mousavian A., Rezvanian A., and Meybodi M. R., Cellular learning automata based algorithm for solving minimum vertex cover problem, in 2014 22nd Iranian Conference on Electrical Engineering (ICEE), 2014, 996–1000.

[10] Beigy H. and Meybodi M. R., utilizing distributed learning automata to solve stochastic shortest path problems, Int. J. Uncertainty, Fuzziness Knowledge-Based Syst., 14(5), Oct. 2006, 591–615

[11] Broder A. *et al.*, Graph structure in the Web, Comput. Networks, 33(1), Jun. 2000, 309–320.

[12] Khomami M. M. D, Bagherpour N., Sajedi H., and Meybodi M. R., A new distributed learning automata based algorithm for maximum independent set problem, in 2016 Artificial Intelligence and Robotics (IRANOPEN), 2016, 12–17.