Efficient Deep Web Crawling Using Reinforcement Learning

Lu Jiang, Zhaohui Wu, Qian Feng, Jun Liu, Qinghua Zheng

MOE KLINNS Lab and SKLMS Lab, Xi'an Jiaotong University
No.28, Xianning West Road, Xi'an 710049, P.R.China
roadjiang@yahoo.com, wzh@stu.xjtu.edu.cn, qfeng@stu.xjtu.edu.cn,
liukeen@mail.xjtu.edu.cn, qhzheng@mail.xjtu.edu.cn

Abstract. Deep web refers to the hidden part of the Web that remains unavailable for standard Web crawlers. To obtain content of Deep Web is challenging and has been acknowledged as a significant gap in the coverage of search engines. To this end, the paper proposes a novel deep web crawling framework based on reinforcement learning, in which the crawler is regarded as an agent and deep web database as the environment. The agent perceives its current state and selects an action (query) to submit to the environment according to Q-value. The framework not only enables crawlers to learn a promising crawling strategy from its own experience, but also allows for utilizing diverse features of query keywords. Experimental results show that the method outperforms the state of art methods in terms of crawling capability and breaks through the assumption of full-text search implied by existing methods.

Keywords: Hidden Web, Deep Web Crawling, Reinforcement Learning.

1 Introduction

Deep web or hidden web refers to World Wide Web content that is not part of the surface Web, which is directly indexed by search engines. Studies [1] show deep web content is particularly important. Not only its size is estimated as hundreds of times larger than the so-called surface Web, but also it provides users with high quality information. However, to obtain such content of deep web is challenging and has been acknowledged as a significant gap in the coverage of search engines [2]. Surfacing is a common solution to provide users deep web content search service[1], in which the crawler pre-computes the submissions for deep web forms and exhaustively indexes the response results off-line as other static HTML pages. The approach enables leveraging the existing search engine infrastructure hence adopted by most of crawlers, such as HiWE (Hidden Web Exposer) [3], Hidden Web crawler [4] and Google's Deep Web crawler [2].

One critical challenge in surfacing approach is how a crawler can automatically generate promising queries so that it can carry out efficient surfacing. The challenge

[1] We may use crawl and surface interchangeably in the rest of the paper.

has been studied by several researches such as [2], [4], [5], [6], [7]. In these methods, candidate query keywords are generated from the obtained records, and then their harvest rates, i.e. the promise to obtain new records, are calculated according to their local statistics, such as DF (Document Frequency) and TF (Term Frequency). The one with the maximum expected harvest rate will be selected for the next query. Their basic idea is similar while the difference is that they choose different strategies to derive the estimated harvest rate of each query candidate.

However, to the best of our knowledge, existing methods suffer from the following three deficiencies. Firstly, the future reward of each query is ignored, which is also known as "myopia problem" [8]. The next query is selected according to the harvest rate defined as the immediate reward at current step. This inherent deficiency makes the existing methods fail to look ahead to future steps thus cannot make a decision of long-term interest. Secondly, the existing methods solely utilize the statistic of acquired data records while ignoring the experience gained from previous queries, which usually results in reducing the efficiency of crawling. For example when a crawler issues an unpromising keyword which brings few or even no response records, as the acquired data record hardly accumulates, the statistic of the data will remain the same. Therefore, it is likely that the crawler will make the same mistakes in its future decisions. Finally, existing methods relied on a critical assumption that full-text search are provided by deep web databases, in which all of the words in every document are indexed. However, this assumption is too strong to hold in the cases of databases providing non full-text search interfaces, in which some words appearing on the pages e.g. noisy words and template words are excluded from the index. The disagreement leads to that the existing estimation techniques for full-text databases, e.g. Zipf' Law [4], [9] can hardly be applied to non full-text databases. E.g. Fig. 1 illustrates the keywords distribution on AbeBooks (www.abebooks.com), in which x-axis represents document frequency ranks of keywords in the data corpus and y-axis denotes the percent of response records brought by the keywords. As one can see, the Zipf curve fails to simulate the distribution of keyword response.

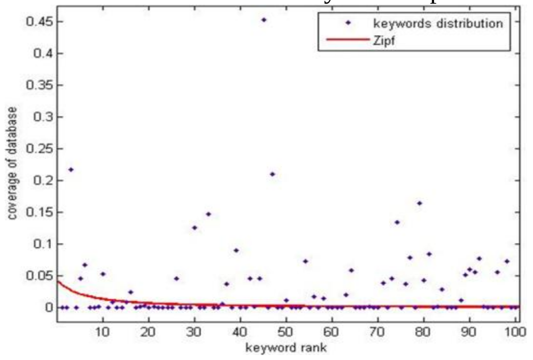


Fig. 1 Keywords distribution on AbeBooks

In this paper, we present a formal framework based on the RL (Reinforcement Learning) [10] for deep web crawling. In the framework, a crawler is regarded as an agent and deep web database as the environment. The agent perceives its current state and selects an action (query) to submit to the environment according to long-term reward. The environment responds by giving the agent some reward (new records) and changing it into the next state. Each action is encoded as a tuple using its linguistic, statistic and HTML features. The rewards of unexecuted actions are evaluated by their executed neighbors. Because of the learning policy, a crawler can avoid using unpromising queries, as long as some of them have been issued. The experimental results on 5 real world deep web sites show that the reinforcement learning crawling method relaxes the assumption of full-text search and outperforms existing methods. To sum up, the main contributions of our work are:

1) We introduce a formal framework for the deep web surfacing problem. To the best of our knowledge, ours is the first work that introduces machine learning approaches to the deep web surfacing problem.
2) We formalize the problem in the framework and propose an efficient and applicable surfacing algorithm working well on both full-text and non full-text databases.
3) We develop a Q-value approximation algorithm allows for a crawler selecting a query according to the long-term reward, which overcomes the myopia problem to some extent.

The rest of this paper is organized as follows: Section 2 gives a brief introduction of related work. Section 3 presents the formal reinforcement learning framework. Section 4 discusses the crawling algorithm and key issues in it. The experimental results are discussed in Section 5 whereas conclusions and future work are presented in the final section.

2 Related Work

The state-of-the-art deep web crawling approaches generate new keywords by analyzing statistic of current acquired records returned from previous queries. Barbosa L. et al. first introduced the ideas, and presented a query selection method which generated the next query using the most frequent keywords in the acquired records [5]. However, queries with the most frequent keywords in hand do not ensure that more new records are returned from the deep web database. Ntoulas A. et al. proposed a greedy query selection method based on the expected harvest rate [4]. In the method, the one with the maximum expected harvest rate will be selected for the next query. Ping W. et al. modeled each web database as a distinct attribute-value graph and a greedy link-based query selection method was proposed to approximate the optimal solution [8]. Lu J. et al. resolved the problem using set-covering sampling method [7]. Liu J. et al. extended the Ntoulas's method to entire form by introducing a novel concept MEP (Minimum Executable Pattern). In the method, a MEP set is build and then promising keywords are selected by joint harvest rate of a keyword and its pattern. By selecting among multiple MEPs, the crawler achieves better results [6]. Jayant M. et al. improved the keyword selection algorithm by ranking keywords by their TFIDF (Term Frequency Inverse Document Frequency) [2]. Jiang L. et al. present a method to evaluate a keyword by its HTML features besides TF and DF using supervised learning [11].

3 Reinforcement Learning Framework

In this section, we propose a formal framework for the deep web crawling based on RL and formalize the crawling problem under the framework. First of all we give an overview of the RL framework.

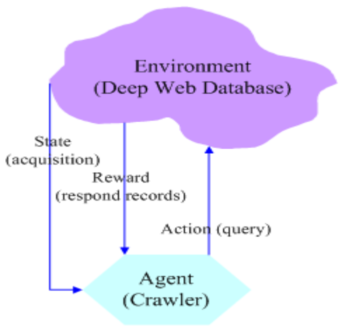


Fig. 2 Overview of the reinforcement learning framework

The relation between a crawler and a deep web database is illustrated in Fig. 2. From the figure, one can conclude that at any given step, an agent (crawler) perceives its state and selects an action (query). The environment responds by giving the agent some (possibly zero) reward (new records) and changing the agent into the successor state. More formally we have

Definition 1. Suppose S and A are two sets of states and actions respectively. A state $s_t \in S$ represents the acquired portion of the deep web database records at the step t. An action $a(k) \in A$ (a for short) denotes a query to the deep web database with the keyword k, which causes a transition from state s_t to some successor state s_{t+1} with the probability $p(s_{t+1}|a, s_t)$.

Definition 2. The process of deep web crawling is defined as a discrete Decision Process (S, A, P) consisting of a set of states S, a set of actions A and transition probabilities distribution P. A crawling process follows a specific issue policy $\pi : S \rightarrow A$, which is a mapping from the set of states to the set of actions.

In the paper, we assume that the decision process is a deterministic process i.e. P is subjected to a uniform distribution. During the process, after execution of an action the agent is responded by giving a collection of data records by the environment. The response record can be defined as:

Definition 3. Suppose D is the collection of all data records residing in deep web database. After execution of action a at state s_t, the response record set $R(s_t, a) \subseteq D$ represents the collection of data records responded by the environment. Likewise the portion of the new records in the response record set retrieved by action a at state s_t is denoted as $R_{new}(s_t, a)$ ($R_{new}(s_t, a) \subseteq R(s_t, a)$).

Suppose a crawling process follows an issue policy π, the portion of new records in the response records of action a at state s_t can be formulated as

$$R_{new}(s_t, a) = R(s_t, a) \backslash \bigcup_{i=1}^{t-1} R(s_i, \pi(s_i)) \quad \textbf{(1)}$$

Note that the response record set of an action is irrelevant to the state hence $\forall i, j$, $R(s_i, a) = R(s_j, a)$.

There are two important functions in the process. Transition function $\delta : S \times A \rightarrow S$ denotes the successor state of the given state and action. Reward function $r(s_t, a)$is the reward received at the transition from state s_t to state s_{t+1} by executing action a, i.e. the portion of new records brought by executinga_t, computed from equation

$$r(s_t, a) = |R_{new}(s_t, a)|/|D| \tag{2}$$

Though in some cases $|D|$ is either unknown or cannot be obtained beforehand, the absence of the value does not influence the calculation of the reward as they are relative values to rank actions in the same baseline.

The transition of actions causes a cost. In the paper, the cost is measured in terms of time consumed, i.e.$cost(s_t, a) = t_a + t_r + |R_{new}(s_t, a)|$. $t_a$is the cost of issuing an action and $t_r$is proportional to the average time of handling a response record.

The expectation conditioned on the current state s and the policy π is called state-value function$V^{\pi}(s)$ of state$s$, computed from

$$V^{\pi}(s_t) = \sum_{i=0}^{h} \gamma^i r(s_{t+i}, \pi(s_{t+i})) \tag{3}$$

in which h is referred as the step length and γ is the discount factor. Among all polices, there must exist an optimal policy, noted π^* defined as$V^{\pi^*}(s) \geq V^{\pi}(s)$ ($\forall s \in S$, $\forall \pi$). To simplify notations, we write $V^{\pi^*} = V^*$.

Based on the presentations above, the formal definition of deep web crawling problem can be defined as:

Problem. *Under the constraint*$\sum_{i=0} cost(s_i, a) \leq cost_{MAX}$,$\forall s_i \in S$ *find such policy* $\pi^* \equiv \arg\max_{\pi} V^{\pi}(s_i)$ *that maximizes the accumulative reward value. Here* $cost_{MAX}$ *is the maximum cost constraint.*

4 Algorithm

In this section we discuss how to resolve the deep web crawling problem defined in Section 3. There are two crucial factors in solving the problem i.e. the reward of each action *r* and the reward of an issue policy Q-value. Section 4.1 and 4.2 introduces the methods for the action reward calculation and Q-value approximation respectively. Finally an adaptive algorithm for surfacing deep web is presented in the end of Section 4.2.

4.1 Reward calculation

Before specifying the method for the action reward calculation, we need the definition of the document frequency.

Definition 4. Suppose on the current state s_t and the issue policy π, the document frequency of action $a(k) \in A$ denoted by $DF(s_t, a(k))$ ($DF(s_t, a)$ for short) is the number of documents containing keyword k in acquired record set $\bigcup_{i=1}^{t-1} R(s_i, \pi(s_i))$.

Note that the document frequency of each action is a known statistic. Since records of $\bigcup_{i=1}^{t-1} R(s_i, \pi(s_i))$ having been retrieved at the step t, the number of documents containing keyword k can be counted up in the acquired record set. Relying on Def. 4, the following theorem can be established.

Theorem 1. At state s_t, the reward of each action a in A can be calculated from

$$r(s_t, a) = \frac{|R(s_t, a)| - DF(s_t, a)}{|D|}. \tag{4}$$

Proof: By incorporating Eq. (1) into Eq. (2) we have

$$r(s_t, a) = |R(s_t, a) \setminus \bigcup_{i=1}^{t-1} R(s_i, \pi(s_i))|/|D|. \tag{5}$$

Eq. (5) can be further rewritten as

$$r(s_t, a) = \frac{|R(s_t,a)| - |R(s_t,a) \cap \bigcup_{i=1}^{t-1} R(s_i,\pi(s_i))|}{|D|}. \tag{6}$$

The intersection part in Eq. (6) denotes the collection of documents containing the keyword of action a in the data set $\bigcup_{i=1}^{t-1} R(s_i, \pi(s_i))$. According to the Def. 4 the value equals to the document frequency of the action, i.e.

$$|R(s_t, a) \cap \bigcup_{i=1}^{t-1} R(s_i, \pi(s_i))| = DF(s_t, a) \tag{7}$$

Consequently the Eq. (4) could be proved by incorporating Eq. (7) into Eq. (6).

The absence of D in Eq. (4) does not affect the final result for the same reason described in Section 3. According to Eq. (4) for an executed action, as response record set $R(s_t, a)$ is acquired, the reward can be calculated. In contrast, the reward calculation for an unexecuted action directly through Eq. (4) is infeasible. Nevertheless, the response record set of an unexecuted action can be estimated by generalizing from those executed. Before proceeding any further, we define the action training and candidate set.

Definition 5. Suppose at state s_t, training set Tr is a set of executed actions, $|Tr| = t$. Similarly, candidate set C is a set of available action candidates for submission in the current state. Each action in either Tr or C is encoded in the same vector space.

Based on Def. 4, for an action a_i in C, its reward can be estimated as:

$$|\tilde{R}(s_t, a_i)| = \sum_{a_j \in Tr} \kappa(a_i, a_j)|R(s_t, a_j)|. \tag{8}$$

in which $\kappa(a_i, a_j)$ is a kernel function used to evaluate the distance between the given two actions. Since the response record set $R(s_x, a)$ of an action is irrelevant to the state s_x, the response record set can be rewritten to the current state s_t i.e. $R(s_x, a) = R(s_t, a)$. Accordingly, the intuition behind the Eq. (8) is to estimate the reward for an action in C by evaluating those in Tr. As all response record sets of executed action are at the current state, the size of response record set of an action in C can be learnt from those sharing the similar features in Tr. Once the size of response record set of an unexecuted is calculated by Eq. (8), the value can then be applied in Eq. (4) to calculate its reward.

Now the action rewards for both executed and unexecuted actions can be calculated from Eq. (4). In the rest of the subsection, we will discuss how to calculate kernel function $\kappa(a_i, a_j)$ in Eq. (8). Calculating the similarity of actions requires encoding them in a feature space. We incorporate three types of features i.e. linguistic features, statistical features and HTML features to establish the feature space [11].

Linguistic features consist of POS (Part of Speech), length and language of a keyword (action). Length is the number of characters in the keyword. Language represents the language that a keyword falls into. It takes effect in multilingual deep web database.

Statistical features include TF (Term Frequency), DF (Document Frequency) and RIDF (Residual Inverse Document Frequency) of a keyword in the acquired records. The value of RIDF is computed as:

$$RIDF = \log(1-e^{-TF/|D|})-\log(DF/|D|) \qquad \textbf{(9)}$$

RIDF tends to highlight technical terminology, names, and good keywords and to exhibit nonrandom distributions over documents [12].

The HTML format usually plays an important role in indicating the semantics of the presented data. This brings us to consider the HTML information of keywords. We propose two HTML features tag-attribute and location. Tag-attribute feature encodes HTML tag and attribute information of a keyword, and location represents the depth of the keyword's node in the DOM tree derived from the HTML document. The features may imply the semantic information of a keyword hence is useful in distinguishing unpromising keywords.

For linguistic and HTML features whose values are discrete, the liner kernel is assigned. Considering that value of statistical feature tends to a Gaussian distribution over documents, the Gaussian kernel is adopted to evaluate similarity upon the statistical features, which is formulated as

$$\kappa_s(a_i, a_j)=exp(-\|a_i - a_j\|^2/2\delta^2). \qquad \textbf{(10)}$$

The final kernel function is hybrid of these kernels. Suppose λ_l, λ_h and λ_s ($\lambda_l+\lambda_h+\lambda_s=1$) are weights for linguistic, HTML and statistical kernel respectively, the kernel function to evaluate similarity of two actions is

$$\kappa = \lambda_l\kappa_l + \lambda_h\kappa_h + \lambda_s\kappa_s. \qquad \textbf{(11)}$$

In experiments the weight of statistical features usually accounts for a larger part.

4.2 Q-value approximation and Surfacing Algorithm

Once the reward of each action is obtained, given the problem definition in Section 3, the agent can find an optimal policy V^* if the V^π of each state can be calculated. The calculation of V^π could be well solved when the agent uses Q-function [13], [14]:

$$V^\pi(s_t) = Q(s_t, \pi(s_t)) = r(s_t, \pi(s_t))+\gamma \max_\pi[V^\pi(\delta(s_t, \pi(s_t)))]. \qquad \textbf{(12)}$$

Here Q-function $Q(s,a)$ represents the reward received immediately upon executing action a from state s, plus the value discounted by γ thereafter. Using Eq. (3), we can rewrite Q-function as

$$Q(s_t,a)=r(s_t,a)+\max[\sum_{i=1}^{h}\gamma^i\times r(s_{t+i},\pi(s_{t+i}))]. \tag{13}$$

To simplify the notion, here we let $\gamma=1$ i.e. the reward for the future steps are regarded as important as those for the present. h is a critical parameter denoting the step length looking ahead to the future reward. If $h=0$, the future reward is ignored and Q-value equals to the immediate reward i.e. $Q(s_t,a)=r(s_t,a)$. When $h>1$, Q-value represents the long-term reward. However, as the action reward at state s_{t+1} is unavailable at state s_t, the Q-value has to be approximated. To estimate the Q-value, we make the following assumption: assume at the current state, the action set A will not enlarge in the next $h+1$ steps ($h\ll|A|$). When h is not very large the assumption is reasonable. Under the assumptions, Theorem 2 could be established.

Theorem 2. At state s_t when $h=1$ the Q-value of an action a_i ($a_i,a_j\in C$,$a_i\neq a_j$) can be estimated as:

$$Q(s_t,a_i)\propto r(s_t,a_i)+\max_j[r(s_t,a_j)+\frac{DF(s_t,a_j)}{|D|}-\frac{|\bigcup_{i=1}^{t}R(s_i,\pi(s_i))||R(s_t,a_j)|}{|D|^2}] \tag{14}$$

Proof: To simplify the notion, let $\bigcup_{i=1}^{t-1}R(s_i,\pi((s_i))=R_{t-}$, $R(s_t,a_i)=R_t$, $R(s_{t+1},a_j)=R_{t+1}$. First of all, because the action set will not enlarge, the optimal Q-value can be searched in the action set at the current state. According to the Eq. (13), when *h* =1 the Q-value can be formulated as

$$Q(s_t,a_i)=r(s_t,a_i)+\max_j[r(s_{t+1},a_j)]. \tag{15}$$

Following the method described in Section 4.1, $r(s_t,a_i)$ can be calculated; whereas $r(s_{t+1},a_j)$is unknown at state s_t. Therefore rewrite the Eq. (15) as

$$Q(s_t,a_i)=\max[|(R_t\cup R_{t+1})\setminus R_{t-}|/|D|]. \tag{16}$$

Because the response records are independent with each others, the capture-mark-recapture [15] method can be applied to estimate the overlaps records:

$$|(R_{t-}\cup R_t)\cap R_{t+1}|/|R_{t+1}|\propto|R_{t-}\cup R_t|/|D|. \tag{17}$$

Further Eq. (17) can be transformed into

$$|R_{t+1}\cap R_t|-|R_{t+1}\cap R_t\cap R_{t-}|\propto|R_t\cup R_{t-}||R_{t+1}|/|D|-|R_{t+1}\cap R_{t-}|. \tag{18}$$

By incorporating Eq. (18) into Eq. (16) we have

$$Q(s_t,a_i)=1/|D|\times\max[|R_t\setminus R_{t-}|+|R_{t+1}\setminus R_{t-}|+|R_{t+1}\cap R_{t-}|-|R_t\cup R_{t-}||R_{t+1}|/|D|] \tag{19}$$

Note that according to the characteristics of response record set in Def. 3 and Eq. (5):

$$|R(s_{t+1},a_j)\setminus R_{t-}|=|R(s_t,a_j)\setminus R_{t-}|=r(s_t,a_j)\times|D|. \tag{20}$$

Following Eq. (5) and Eq. (20), Eq. (19) can be reformulated as

$$Q(s_t,a_i)=r(s_t,a_i)+\max_j[|R_{t+1}\cap R_{t-}|/|D|+r(s_t,a_j)-|R_t\cup R_{t-}|\times|R_{t+1}|/|D|^2] \tag{21}$$

Then the Theorem 2 can be derived by incorporating Eq. (7) into Eq. (21).

As all the factors in the Eq. (14) are either calculated or can be statistically numerated, the Q-value of each action can be approximated based on the acquired data set. Now, we can approximate Q-value with a given step length by iteratively applying Eq. (14). Due to the lack of space, we cannot present the details here. Note if *h* goes too big, the assumption may not hold and the future state may diverge from the experienced state rendering the approximation for future reward imprecise.

We develop an adaptive algorithm for deep web surfacing based on the framework, as shown in Algorithm 1. The algorithm takes the current state and last executed action as input and outputs the next optimal action.

Algorithm 1: Adaptive RL surfacing algorithm

```
Input: $s_t,\ a_m$        Output: $\pi(s_{t+1})$
1:  calculate the reward of action $a_m$ following Eq.(4);
2:  for each document $d_i \in R(s_{t-1}, a_m)$
3:      for each keyword $k$ in $d_i$ do
4:          if action $a(k) \notin A$ then $A = A \cup a(k)$;
5           else then update TF and DF of action $a(k)$;
6:      end for
7:  end for
8:  change the current state to $s_{t+1}$;
9:  $Tr = Tr \cup \{a_m\}$;  update candidate set $C$; $C = C \setminus \{a_m\}$;
10: for each $a_i \in C$ update its reward using Eq. (8) and Eq. (4);
11: for each $a_i \in C$ calculate its Q-value using Eq. (14);
12: return $\arg\max_a [Q(s_t, a)]$;
```

Specifically, the surfacing algorithm first calculates the reward of the last executed action and then updates the action set through Step 2 to Step 7, which causes the agent to transit from its state s_t to the successor state s_{t+1}. Then the training and candidate set are updated in accord with the new action set in Step 9. After that the algorithm estimates the reward and Q-value for each action in candidate set in Step 10 and Step 11 respectively. The action that maximizes Q-value will be returned as the next to be executed action.

5 Experiments

To demonstrate the efficiency of our proposed approach for deep web crawling, we execute our algorithm on five real world deep web databases with different scales and domains. The detailed information about these databases is listed in Tab.1. In the case of AbeBooks, as its large scale, the agent restricted to crawling the "historical fictions" category to accelerate the experiment. Queries to the site are applied to the textbox "keywords". Regarding Wikicfp, Yahoo movie, which are the typical medium Deep Web databases, we utilize the only generic search textboxes as their query interfaces. As for Baidu Baike and Google Music, the sites are multilingual sites consisting of both English and Chinese. On the sites we select the rewarding textbox "Keyword Tag" and "Singer" as their query interfaces.

To compare our RL method with existing ones, we choose following three methods as baseline methods: (Suppose at state s_t, the agent is to evaluate an action a)

- Random [4], [7], [8]: the reward of an action is assigned to a random float i.e. $Q(s_t, a) = random(1.0)$.
- GF (Generic Frequency) [5], [7], [8]: the reward of an action is evaluated by the generic DF of the action at current state, i.e. $Q(s_t, a) = DF(s_t, a)$.
- Zipf [4], [6], [9]: The size of response record set of each action is estimated by Zipf-Mandelbrot's Law [16]: $|R(s_t, a)| = \alpha(r+\beta)^{-\gamma}|$,where α, β and γ are parameters and r is the DF rank of the action.

All methods above (including RL) share the same candidate set generation policy which selects the top 500 actions with highest RIDF. It is interesting to note that RL is more general compared with the baseline methods. If future reward of an action is ignored i.e. $h = 0$ and the reward of an action is determined by a presumed distribution, the RL degenerates to Zipf, i.e. $Q(s_t, a_i) = r(s_t, a_i)$. Further if the acquired portion of an action is ignored too i.e. $R(s_t, a_i) = \Phi$, the RL degenerates to the GF, i.e. $Q(s_t, a_i) = DF(s_t, a_i)$.

5.1 Effectiveness of RL method

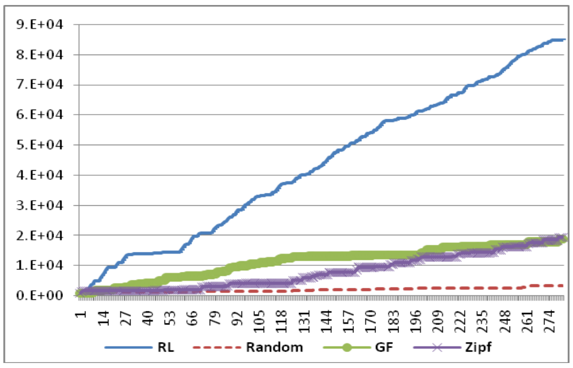
Our interest is to discover their records as many as possible with affordable cost. To make the results more intelligible, we roughly use *harvest* (Fourth column) i.e. the number of actual retrieved records and *number of queries* (Sixth column) to evaluate the crawling effects. Tab. 1 shows that our method is quite efficient. In the first four cases the agent achieves more than 80% coverage by issuing around 500 hundred queries.

Table 1. Experiment web sites and their experimental results.

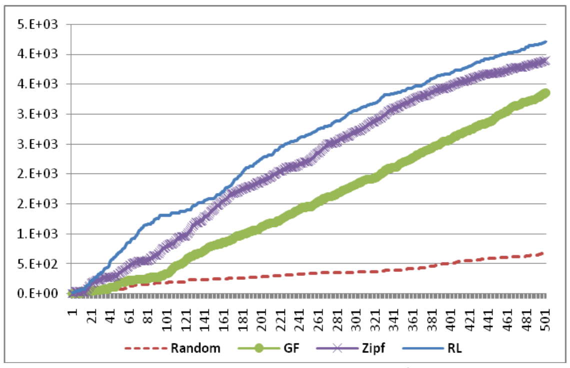
DB Name	URL	Domain	Harvest	Estimated Database	#Queries
AbeBooks	www.abebooks.com	Book	90,224	110,000	322
Wikicfp	www.wikicfp.com	Conference	4,125	5,200	499
Yahoo movie	movies.yahoo.com/mv/search	Movie	126,710	128,000	367
Google music	www.google.cn/music/	Music	85,378	110,000	592
Baidu Baike	Baike.baidu.com	Wikipedia	820,180	1,000,000	1,950

5.2 Performance Comparison with baseline method

We performed our method as well as the baseline methods on the experiment sites. The experimental results are displayed in Fig. 3 in which the y-axis denotes the database coverage, while the x-axis represents the query number (due to the lack of space we only present some results here). In experiments step length was set to 1. As can be seen, the result shows that RL method is more efficient than baseline methods on the experiment websites. We analyzed the queries logs and summarized two reasons accounting for excellence of RL method. Firstly because the RL method selects a keyword according to the long-term rather than immediate reward, it is able to acquire a better awareness of the environment leading to more accurate estimation for succeeding rewards. As we found in experiments the rewarding keywords are issued earlier in RL than other methods. Secondly the keywords issued in RL are more relevant to the pattern selected, e.g. "keywords" on Baidu Baike. This suggests that the agent using RL learns the experience from its previous queries and hence sticks on the keywords matching against more records; whereas the agent using other methods do not make any adjustment when the presumed assumption is not applied.



(a) Experiment on Baidu Baike



(b) Experiment on Wikicfp

Fig. 3 Performance Comparisons with baseline methods

6 Conclusion and future work

In this paper we tackle the problem of deep web surfacing. The paper first presents a formal reinforcement learning framework to study the problem and then introduce an adaptive surfacing algorithm based on the framework and its related methods for reward calculation and Q-value approximation. The framework enables a crawler to learn an optimal crawling strategy from its experienced queries and allows for it making decisions on long-term rewards. Experimental evaluation on 5 real deep web sites suggests that the method is efficient and applicable. It excels the baseline method and works well on both full-text and non full-text databases. In general, it retrieves more than 80% of the total records by issuing a few hundreds of queries.

We are studying the issues of deep web crawling in practical and developing an open source platform for Deep Web crawling: DWIM (Deep Web Intelligent Miner). DWIM is the first open source software in the respect Deep Web crawling and integrates many crawling policies and keywords selection criteria which can be used as an experimental platform for researches and a crawler engine for developers.

Acknowledgement

The research was supported by the National High-Tech R&D Program of China under Grant No.2008AA01Z131, the National Science Foundation of China under Grant Nos.60825202, 60803079, 60633020, the National Key Technologies R&D Program of China under Grant Nos.2008BAH26B02, 2009BAH51B00, the Open Project Program of the Key Laboratory of Complex Systems and Intelligence Science, Institute of Automation, Chinese Academy of Sciences under Grant No. 20080101, Cheung Kong Scholar's Program.

References

1. Lawrence, S., Giles, C.L.: Searching the World Wide Web. Science, pp. 280:98--100 (1998)
2. Madhavan, J., Ko, D., Kot, L., Ganapathy, V., Rasmussen, A., Halevy, A.: Google's Deep-Web Crawl. In Proceedings of VLDB2008. Auckland, New Zealand, pp. 1241--1252 (2008)
3. Raghavan, S., Garcia-Molina, H.: Crawling the Hidden Web. In Proceedings of VLDB2001. Rome Italy, pp. 129--138 (2001)
4. Ntoulas, A., Zerfos, P., Cho, J.: Downloading Textual Hidden Web Content through Keyword Queries. In Proceedings of JCDL2005. Denver, USA. pp. 100--109 (2005)
5. Barbosa, L., Freire, J.: Siphoning Hidden-Web Data through Keyword-Based Interfaces. In Proceedings of SBBD2004, Brasilia, Brazil, pp. 309--321 (2004)
6. Liu, J., Wu, ZH., Jiang, L., Zheng, QH., Liu, X.: Crawling Deep Web Content Through Query Forms. In Proceedings of WEBIST2009, Lisbon Portugal, pp. 634--642 (2009)
7. Lu, J., Wang, Y., Liang, J., Chen, J., Liu J.: An Approach to Deep Web Crawling by Sampling. In Proceedings of IEEE/WIC/ACM Web Intelligence, Sydney, Australia, pp. 718--724 (2008)
8. Wu, P., Wen, JR., Liu, H., Ma, WY.: Query Selection Techniques for Efficient Crawling of Structured Web Source. In Proceedings of ICDE2006, Atlanta GA, pp. 47--56 (2006)
9. Ipeirotis, P., Gravano, L.: Distributed search over the hidden web: Hierarchical database sampling and selection. In VLDB2002, Hong Kong, China, pp.394--405 (2002)
10. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: A survey. Journal of Artificial Intelligence Research, 4:237–285 (1996)
11. Jiang, L., Wu ZH., Zheng, QH., Liu, J.: Learning Deep Web Crawling with Diverse Features. In Proceedings of IEEE/WIC/ACM Web Intelligence, Milan, Italy, pp. 572--575 (2009)
12. Yamamoto, M., Church, K.W.: Using suffix arrays to compute term frequency and document frequency for all substrings in a corpus. Computational Linguistics, 27(1): pp. 1-30 (2001)
13. Watkins, C.J., Dayan, P.: Q-learning. Machine Learning, 8:279-292 (1992)
14. Ratsaby, J.: Incremental Learning with Sample Queries, IEEE Trans. on PAMI, Vol. 20, No. 8, pp.883-888 (1998)
15. Amstrup, S.C., McDonald, T.L., Manly, B.F.J.: Handbook of capture–recapture analysis. Princeton University Press (2005)
16. Mandelbrot, B.B.: Fractal Geometry of Nature. New York: W. H. Freeman and Company (1988)
17. Sutton, R.C., Barto, A.G.: Reinforcement learning: An Introduction. The MIT Press, Cambridge, MA (1998)