CRAWLING DEEP WEB CONTENT THROUGH QUERY FORMS

Jun Liu, Zhaohui Wu, Lu Jiang, Qinghua Zheng and Xiao Liu
MOE KLINNS Lab and SKLMS Lab, Xi’an Jiaotong University, Xi’an 710049, China

Keywords: Deep Web, Deep Web Surfacing, Minimum Executable Pattern, Adaptive Query.

Abstract: This paper proposes the concept of Minimum Executable Pattern (MEP), and then presents a MEP generation method and a MEP-based Deep Web adaptive query method. The query method extends query interface from single textbox to MEP set, and generates local-optimal query by choosing a MEP and a keyword vector of the MEP. Our method overcomes the problem of “data islands” to a certain extent which results from deficiency of current methods. The experimental results on six real-world Deep Web sites show that our method outperforms existing methods in terms of query capability and applicability.

1 INTRODUCTION

There is an enormous amount of information buried in the Deep Web, and its quantity and quality are far beyond the “surface web” that traditional search engines can reach (Michael, 2001). However, such information cannot be obtained through static html pages but dynamic pages generated in response to a query through a web form. Due to the enormous volume of Deep Web information and the heterogeneity among the query forms, effective Deep Web crawling is a complex and difficult issue.

Deep web crawling aims to harvest data records as many as possible at an affordable cost (Barbosa, 2004), whose key problem is how to generate proper queries. Presently, a series of researches on Deep Web query has been carried out, and two types of query methods, namely prior knowledge-based methods and non-prior knowledge methods, have been proposed.

The prior knowledge-based query methods need to construct the knowledge base beforehand, and generate queries under the guidance of prior knowledge. In (Raghavan, 2001) proposed a task-specific Deep Web crawler and a corresponding query method based on Label Value Set (LVS) table; the LVS table as prior knowledge is used for passing values to query forms. (Alvarez, 2007) brought forward a query method based on domain definitions which increased the accuracy rate of filling out query forms. Such methods automate deep crawling to a great extent (Barbosa, 2005), however, have two deficiencies. First, these methods can only perform well when there is sufficient prior knowledge, while for the query forms that have few control elements (such as a single text box), they may have an unsatisfactory performance. Second, each query is submitted by filling out a whole form, which reduces the efficiency of Deep Web crawling.

The non-prior knowledge methods are able to overcome the above deficiencies. These methods generate new candidate query keywords by analyzing the data records returned from the previous query, and the query process does not rely on prior knowledge. Barbosa et al. first introduced the ideas, and presented a query selection method which generated the next query using the most frequent keywords in the previous records (Barbosa, 2004). However, queries with the most frequent keywords in hand do not ensure that more new records are returned from the Deep Web database. (Ntoulas, 2005) proposed a greedy query selection method based on the expected harvest rate. In the method, candidate query keywords are generated from the obtained records, and then their harvest rates are calculated; the one with the maximum expected harvest rate will be selected for the next query. (Wu P, 2006) modeled each web database as a distinct attribute-value graph, and under this theoretical framework, the problem of finding an optimal query selection was transferred into finding a Weighted Minimum Dominating Set in the corresponding attributed-value graph; according to the idea, a greedy link-based query selection method was proposed to approximate the optimal solution.
Compared with the prior knowledge-based methods, the non-prior knowledge methods improve the query capability on Deep Web crawling. However, these methods suffer from the following three limitations: first, queries are only based on single text box and the candidate keywords are assumed to be suited to the text box; second, query selection decision is made solely based on the obtained records, namely “myopia estimation problem” (Wu P, 2006); third, query selection lacks sufficient knowledge in the initial period. The three problems limit the query capability on Deep Web crawling, and result in the phenomenon called “data islands” which means the total acquired records may constitute only a small fraction of the target database.

In this paper, we propose the concept of Minimum Executable Pattern (MEP) and a MEP-based Deep Web adaptive query method. The query method extends query interface from single textbox to MEP set; it performs a query by choosing a MEP and a keyword vector of the MEP, and generates the next query with the maximum expected efficiency adaptively through the acquired information. This method not only has the advantages over the non-prior knowledge methods, but also has the ability to solve the problem of “data islands” by making the most of query capability of elements in MEP set. The experimental results on six real-world Deep Web sites show that our method outperforms existing methods in terms of query capability and applicability.

The rest of the paper is organized as follows: Section 2 introduces the concept of MEP. Section 3 describes the algorithm for generating the MEP set of a given query form. Section 4, the core of this paper, studies the MEP-based adaptive query method and the convergence of the related algorithm. The experimental results are discussed in Section 5, and conclusions and future work are offered in the final section.

2 FUNDAMENTAL CONCEPTS

Definition 1: Query Form. A query form $F = \{e_1, \ldots, e_n\}$, where $e_i$ is an element of $F$, such as a checkbox, text box or radio button.

Each element $e_i$ of $F$ has its domain $D_i$ which is the set of values associated with $e_i$. If $D_i$ is a finite set, then $e_i$ is a finite domain element, else $e_i$ is an infinite domain element. Elements are usually associated with some descriptive text to help users understand the semantics of the element, namely labels. The expression $\text{label}(e_i)$ is used to denote the label of $e_i$ (Raghavan, 2001).

Definition 2: Executable Pattern (EP). Given a query form $F = \{e_1, \ldots, e_n\}$, $\forall \{e'_1, \ldots, e'_m\} \subseteq 2^F, 1 \leq m \leq n$, $\{e'_1, \ldots, e'_m\}$ is an executable pattern of $F$ if the Deep Web database returns the corresponding results after the query, with value assignments of elements in $\{e'_1, \ldots, e'_m\}$ is issued.

Executable Pattern satisfies the following monotonicity properties:

1. If $\{e'_1, \ldots, e'_m\}$ is an executable pattern, then any subset of $F$ that contains $\{e'_1, \ldots, e'_m\}$ is also an executable pattern.
2. If $\{e'_1, \ldots, e'_m\}$ is not an executable pattern, then any subset of $\{e'_1, \ldots, e'_m\}$ is not an executable pattern.

The result records returned based on an executable pattern may also be null. A query based on a non-executable pattern cannot perform a successful database search, and usually leads to an error report or switches to another page.

Definition 3: Minimum Executable Pattern (MEP). If $\{e'_1, \ldots, e'_m\}$ is an executable pattern of query form $F = \{e_1, \ldots, e_n\}$ $(1 \leq m \leq n)$, then $\{e'_1, \ldots, e'_m\}$ is a MEP iff any proper subset of $\{e'_1, \ldots, e'_m\}$ is not an executable pattern. We may rewrite it as $\text{MEP}(e'_1, \ldots, e'_m)$.

The keyword vector $kv = (kv_1, \ldots, kv_m)$ makes a value assignment for $\text{MEP}(e'_1, \ldots, e'_m)$, where $kv_i \in D'_i$, $i = 1, 2, \ldots, m$. If there is an infinite set $D'_i$, then the MEP is called infinite domain MEP, or IMEP for short; while if each $D'_i$ is a finite set, then the MEP is called finite domain MEP (FMEP). All MEP of the query form $F$ constitute the MEP set of $F$ which is denoted as $S_{\text{MEP}}$.

According to the monotonicity properties of EP, we can draw the following inference:

Inference 1: An executable pattern $\{e'_1, \ldots, e'_m\}$ is a MEP iff all its subsets of size $m-1$ are not EP.

Deep web crawling aims to retrieve data records from a web database through iterative queries on a given query form $F$. In this process, the MEP set
\( S_{\text{MEP}} \) of the query form \( F \) is generated firstly and then all subsequent queries are performed based on the MEP set. For this reason, our study focuses on such two critical issues: first, how to generate the MEP set of a given query form; second, how to select a proper MEP and its keywords vector to harvest data records from web database efficiently.

### 3 MEP SET GENERATION

A naive method to generate the MEP set is to enumerate all combinations of elements in the form \( F \). If the size of \( F \) is \( n \), then the number of combinations to be checked totals is about \( 2^n \). For this reason, the efficiency of enumeration will sharply drop when \( n \) grows large.

Elements in a form are not independent but always have connection with each other, such as “start city” and “destination city” in a ticket query form. Such elements always appear simultaneously in the same MEP, and their combination is called “Condition Pattern” (CP for short) (Zhang Z, 2004). By using CP, the MEP set can be generated with greater granularity than element, which greatly improves the efficiency of MEP set generation.

There have existed several methods of generating all CPs of a query form (He B, 2006) (Zhang Z, 2004). For example, Zhang Z. et al. proposed the 2P grammar & best-effort parser model, by using which, a query form can be parsed into a complete parser tree (Zhang Z, 2004), and the CP nodes in this tree are corresponding with the CPs of the form. The CP set can be easily generated through finding all the CP nodes in this tree.

Let the CP set be \( S_{\text{CP}} \), and the initial MEP set \( S_{\text{MEP}} \) is empty. The algorithm MEPGeneration\((S_{\text{CP}}, S_{\text{MEP}})\) will generate the MEP set \( S_{\text{MEP}} \) based on the CP set \( S_{\text{CP}} \). In order to facilitate the description of the algorithm, we introduce a function \( \mu \) defined as: 
\[
\mu(A) = \{ A \cap \{ x \mid x \in A \} \} (A \neq \emptyset).
\]

The algorithm is shown in Fig.1.

**Figure 1:** Algorithm for generating the set of MEP.

\[
\begin{align*}
\text{MEPGeneration}(S_{\text{CP}}, S_{\text{MEP}}) \\
& \text{Step 1: If none of } \mu(S_{\text{CP}}) \text{ is EP} \\
& \quad \text{Add } S_{\text{CP}} \text{ to } S_{\text{MEP}}; \quad \text{Return;} \\
& \text{Step 2: Else for each } S'_{\text{CP}} \in \mu(S_{\text{CP}}) \text{ that is EP} \\
& \quad \text{MEPGeneration}(S'_{\text{CP}}, S_{\text{MEP}}); \\
& \text{Step 3: Return } S_{\text{MEP}}.
\end{align*}
\]

According to the monotonicity properties of EP, the algorithm uses divide-and-conquer to generate \( S_{\text{MEP}} \) in a recursive way. Considering that a single CP is also a MEP in most cases, we can move all CPs that are EPs from SCP to SMEP before executing the above algorithm, which can accelerate the generation of MEP set.

### 4 QUERY BASED ON MEP

Once the MEP set \( S_{\text{MEP}} \) of a given form is obtained, the next task is to select suitable MEPs and the corresponding keywords vectors to perform iterative queries on the target Deep Web database. In this section, we formulate the problem of MEP-based query, and on the basis of that, propose a MEP-based adaptive query algorithm as well as the estimation methods of two key parameters. Finally, we give the convergence analysis of our query algorithm.

#### 4.1 Formal Description of MEP-based Query

Let \( q(kv, \text{mep}) \) be the \( i \)th query on the target Deep Web site, and \( q(kv, \text{mep}) \) is implemented using the MEP \( \text{mep} \) and its corresponding keyword vector \( kv \). Here, \( \text{mep} \in S_{\text{MEP}} \) includes \( m \) elements and \( kv=(kv_1, kv_2, ..., kv_m) \) is a \( m \)-dimensional vector accordingly. \( q(kv, \text{mep}) \) can be abbreviated as \( q_i \).

Given a query \( q_i \), \( P(q_i) \) is used to denote the fraction of data records returned form the web database through the query \( q_i \). \( P(q_i \lor \ldots \lor q_i) \) represents the fraction of the common records that are returned from \( q_i \), \( q_2 \),..., and \( q_i \). Similarly, we use \( P(q_i \land \ldots \land q_i) \) to represent the fraction of the unique records that are returned from \( q_i \) or \( q_2 \),..., or \( q_i \). Additionally, \( P_{\text{new}}(q_i) \) stands for the fraction of the new records that have not been retrieved from previous queries from \( q_i \). \( P_{\text{new}}(q_i) \) is computed from Equation (1):

\[
P_{\text{new}}(q_i) = P(q_i \lor \ldots \lor q_i) - P(q_i \land \ldots \land q_i) \tag{1}
\]

In order to measure the resource consumption of issued query, we introduce cost\((q_i)\) to represent cost of issuing the query \( q_i \). Depending on the scenario, the cost can be measured either in time, network bandwidth, or the number of interactions with the site. In this study, we measure cost in terms of consumed time, as follows:

\[
\text{cost}(q(kv, \text{mep})) = t_0(\text{mep}) + t_i(\text{q}(q(kv, \text{mep}))) + t_0(\text{P}_{\text{new}}(q(kv, \text{mep}))) \tag{2}
\]
In the above equation, query cost consists of three factors, \( t_q(mep_j) \) is fixed part of the query cost, which includes the query transmitting time and query processing time by Deep Web database; \( t_d \) is proportional to the average time of handling a result record; while \( t_d \) is proportional to the average time of downloading a new result record.

With the above notations, we can formalize MEP-based query on Deep Web as follows: under the constraint \( \sum_{i=1}^{n} \text{cost}(q_i) \leq T \), find a sequence of queries \( q_i, q_j, \ldots, q_n \) that maximize \( P_{\text{new}}(q_i \lor \ldots \lor q_n) \). Here, \( T \) is the maximum cost constraint.

### 4.2 Adaptive Query Algorithm

The sequence of queries \( q_i, q_j, \ldots, q_n \) that maximize \( P_{\text{new}}(q_i \lor \ldots \lor q_n) \) is called global-optimal query set. Even if all results of candidate queries are clearly known, finding the global-optimal query set is an NP-Hard problem. An efficient algorithm to solve this problem in polynomial time has yet to be found. For this reason, we present an adaptive query algorithm based on MEP that aims at finding a local-optimal query set to approximate the global optimal query set. A query is the local-optimal if it has the maximum value of Efficiency. Efficiency is defined as follows:

**Definition 4: Efficiency.** Efficiency \((q_i)\) is used to quantify new queries returned from \( q_i \) per unit cost:

\[
\text{Efficiency}(q_i|kv,mep) = \frac{P_{\text{new}}(q_i|kv,mep)}{\text{cost}(q_i|kv,mep)}
\]

(3)

By observing equation (2) and (3), it can be seen that to compute \( \text{Efficient}(q_i) \) is actually to compute \( P_{\text{new}}(q_i) \). Using chain rules, \( P_{\text{new}}(q_i) \) can be rewritten as:

\[
P_{\text{new}}(q_i|kv,mep) = P_{\text{new}}(q_i|mep) \cdot P_{\text{new}}(q_i|kv)\text{cost}(q_i|kv,mep)
\]

(4)

In equation (4), the value of \( P_{\text{new}}(q_i) \) is determined by a joint decision of both \( P_{\text{new}}(q_i|mep) \) and \( P_{\text{new}}(q_i|kv,mep) \). \( P_{\text{new}}(q_i|kv,mep) \) is also called harvest rate of the \( mep \), (i.e. the capability of obtaining new records), which is independent of choice of keyword vectors, but depends on the pattern \( mep \) itself. For example, assuming that a Deep Web site about academic paper has MEP set \( S_{\text{MEP}} = \{ \text{mep(Keywords)}, \text{mep(Abstract)} \} \), it is obvious that the harvest rate of “Abstract” pattern is greater than that of “Keywords”, since keywords are usually included in abstract. \( P_{\text{new}}(q_i|kv,mep) \) represents the conditional harvest rate of \( kv \) among all candidate keyword vectors of given \( mep \). The value depends on the capability of obtaining new records of selected keyword vector.

The estimation of \( P_{\text{new}}(q_i|mep) \) and \( P_{\text{new}}(q_i|kv,mep) \) is the key to find the local-optimal query in our adaptive query algorithm. The process of estimation is based on currently available records. In the early stage of Deep Web crawling, as feedback records are relatively scarce, the selection of keyword vectors is lack of basis and inevitably leads to the problem of “data islands”. To address the problem, we introduce an LVS table in our algorithm. The algorithm is divided into two phases. When the number of queries is less than a certain threshold \( s \), i.e. in the data accumulation phase, the algorithm uses Probabilistic Ranking Function (Raghavan, 2001), an LVS value assignment method, to select the most promising \( kv \) and obtains data from the target Deep Web database. Once the number of queries is greater than or equal to \( s \), the algorithm switches to the prediction phase. In this phase, it analyzes currently available data and estimates the most promising query. The algorithm finally outputs the next local-optimal query. The algorithm flow is shown in Fig.2.

In the following, the methods for computing \( P_{\text{new}}(q_i|mep) \) and \( P_{\text{new}}(q_i|kv,mep) \) are discussed in details.

#### 4.2.1 Prediction for \( P_{\text{new}}(q_i|mep) \)

In practice, we use \( P_{\text{new}}(q_i|mep) \) to denote the predicted value of \( P_{\text{new}}(q_i|mep) \) at the \( i \)th query and introduce two methods to accomplish the task.

### Figure 2: Adaptive query algorithm based on MEP.

1. **Continuous prediction:** the current harvest rate of a MEP totally depends on the harvest rate of the latest issued query by the MEP, namely:
In order to calculate \( P(q(kv|mep)) \land (q_1 \lor \ldots \lor q_i, j) \), we introduce the notion of SampleDF\(w \), which means the document frequency of observed word \( w \) in sample groups \( \{d_1, \ldots, d_i\} \) (Ipeirotis, 2002).

\[
\text{SampleDF}(w) = \sum_{l=1}^{k} h_l = 1 \text{ if } w \text{ in } d_l, \text{ 0 otherwise.}
\]

Unfortunately, SampleDF\(w \) cannot be applied to our work, for it only focuses on single keyword and ignores the compatibility between \( kv \) and \( mep \). To identify the contribution of the document frequency of \( m \)-dimensional keyword vector on a given particular pattern, we introduce \( \cos(kvx^i, mepx^i) \), where \( kvx^i \) is the corresponding Boolean vector of \( kvx \) in \( d_i \), and similarly \( mepx \) is the Boolean vector of \( mepx \). Assume that cosine value of null vector and any vector is 0. Here, we define the document frequency of \( kvx \) on a given \( mepx \) in sample groups \( \{d_1, \ldots, d_i\} \) as SampleDF\(kvx|mep \), which is calculated as follows:

\[
\text{SampleDF}(kvx|mep) = \sum_{i=1}^{k} \cos(kvx^i, mepx^i) = \sum_{i=1}^{k} \frac{\sum_{x \in \text{SampleDF}(kvx|mep)} (mepx^i \land kvx^i)}{\sum_{x \in \text{SampleDF}(kvx|mep)} mepx^i}
\]

In equation (8), \( mepx = (mepx_1, \ldots, mepx_m, mepx_m) \). Unlike IMP, FMEP keyword vector can be obtained by parsing the form \( F \). In order to eliminate the influence on predicting keyword vector of IMP, we assign zero to \( mepx^i \) of finite element; \( kvx^i = (kvx_{i,1}, \ldots, kvx_{i,m}) \), for given \( kvx \) and \( mepx \). the \( kvx^i \) generation algorithm is shown in fig.3, where label\( (kvx) \) is the keyword label in \( d_i \), label\( (e) \) is the label of the \( e \)th element of the \( mepx \).

The reason of label\( (kvx) \) = null in Step 4 is that either the label of \( kvx \) is absent or the label can’t be extracted. To solve this problem, the algorithm borrows the idea from (Raghavan, 2001) to find the most relevant label of the missing label. In Step 6, \( M_s(x) \) is the fuzzy value of \( x \) in LVS. According to fig.3, the SampleDF\(kvx|mep \) of a given FMEP is 0.
Obviously, when the given \( mep \) is FMEP, all candidate keyword vectors of \( mep \) can be obtained by parsing the form \( F \). Assume that each pattern is capable to obtain all records from the target Deep Web database within limited time of queries, we can simply use average value to estimate \( P(q(kv|mep)) \):

\[
P(q(kv|mep)) = \frac{1}{\prod_{i=1}^{p} |D_i|}
\]

(9)

where \( D_i \) is the domain of element \( e_i \) in \( mep_j \).

Furthermore, suppose the \( mep \) consists of \( p \) radio buttons (or combo boxes) \( e_1, e_2, \ldots, e_p \), and \( q \) checkboxes \( e_{p+1}, e_{p+2}, \ldots, e_{p+q} \), \( e_i \) has \( m_i \) buttons (\(|D_i| = m_i\)) and \( e_{p+1} \) has \( r \) boxes (\(|D_{p+1}| = 2^r \)). Since the domain size of a checkbox is an exponential function of the number of its check buttons, it is difficult to cover the full domain within acceptable time. In order to improve efficiency of queries with checkboxes, we replace the domain of a checkbox to its subset with size of \( h_i (2^{\leq h_i \leq r+2}) \). Experimental results indicate that the subset formed by empty set, full set and sets of an individual checkbox can obtain about 90% of total records in most cases. Based on the above considerations, the value of \( P(q(kv|mep)) \) can be further optimized as:

\[
P(q(kv|mep)) = \frac{1}{\prod_{i=1}^{p} \prod_{h_i} \prod_{m_i}}
\]

(10)

When the given \( mep_j \) is IMEP and consists of \( p \) radio buttons and \( q \) checkboxes, the value of \( P(q(kv|mep)) \) is

\[
P(q(kv|mep)) = \frac{f}{\prod_{i=1}^{p} \prod_{h_i} \prod_{m_i}}
\]

(11)

Compared with equation (10), equation (11) is more general. When the given \( mep \) is FMEP, \( f = 1 \), which means that the query scope is the entire database; when the given \( mep \) is IMEP, \( f \) is the fraction of records including keywords of infinite domain element on the given \( mep_j \). Here, we exploit Zipf-Mandelbrot law to estimate the fraction. Alternative method such as Poisson Estimator (Kenneth, 1995) may also be exploited. Zipf was the first to observe that the word-frequency distribution followed a power law, which was later refined by Mandelbrot. Mandelbrot observed a relationship between the rank \( r \) and the frequency \( f \) of a word in a text database (Mandelbrot, 1988):

\[
f = \alpha (r + \beta)^{-\gamma}, \text{ where } \alpha, \beta \text{ and } \gamma \text{ are parameters.} \tag{12}
\]

By analyzing dozens of experimental results, we find \( kv \) also follows Zipf-Mandelbrot law. If the rank value of \( \text{SampleDF} (kv|mep) \) is known, the value of \( f \) can also be estimated using equation (12). Then equation (11) can be rewritten as follows:

\[
P(q(kv|mep)) = \frac{\alpha(r + \beta)^{-\gamma}}{\prod_{i=1}^{p} \prod_{h_i} \prod_{m_i}}
\]

(13)

Once \( \text{SampleDF} (kv|mep) \) and \( P(q(kv|mep)) \) are calculated, \( P_{\text{new}}(q(kv|mep)) \) can be predicted as follows: a candidate query is formulated as a 4-tuple \( (\text{MEP}, \text{Keyword Vector}, \text{SampleDF}, \text{ActualDF}) \), where \( \text{ActualDF} \) indicates the actual number of records returned by the issued query; all the 4-tuples of candidate queries construct the query candidate pool. Our predicting algorithm manages the query candidate pool based on the acquired data records of the last query, sorts all 4-tuples in the pool according to \( \text{SampleDF} \), and then fits equation (12) with the rank and \( \text{ActualDF} / S \) (the size of the target database) of all tuples that meet \( \text{SampleDF}*\text{ActualDF} \neq 0 \). Subsequently, \( P_{\text{new}}(q(kv|mep)) \) of all the tuples which meet \( \text{ActualDF} = 0 \) can be calculated according to equation (11). The detailed algorithm is shown in Fig.4.

**Figure 4:** Algorithm for predicting \( P_{\text{new}}(q(kv|mep)) \).

---

4.3 Convergence Analysis

When to stop querying the web database is a difficult issue, especially when the size of target database is unknown. To make a decision on “When to stop” requires the knowledge of the relationship between fraction of new records and query cost. Assume that the size of Deep Web database is $S$, $m_i$ is the fraction of records returned by the $i$th query and $a_i$ is the cumulated fraction of new records returned by previous $i$ queries. We have $a_{i+1} = a_i + m_i * p_i$, where $p_i$ is the fraction of new records in $m_i$.

To simplify the calculation, we suppose that $m_i$ is fixed and $p_i$ can be approximated as $1/k - S/a - S$, where $k$ is the number of records that are not retrieved.

The comparison between ideal and practical crawling fraction curve is carried out on dozens of Deep Web databases. Take the Babe Raccoon (http://vod.xjtu.edu.cn) as an example (see Fig.5). Figure 5: Comparison between ideal and practical crawling fraction.

From both ideal and practical fraction curve, we can find that when the fraction of the total records obtained is approaching 90%, the harvested rate of the subsequent queries is relatively small and even equals to 0 in a period of time. We call the phenomenon “Crawler Bottleneck”. Deep web crawling is a time-consuming task requiring a significant amount of network resources. It would be a waste of both time and network bandwidth if the crawler doesn’t stop crawling when it comes into “Crawler Bottleneck” phase. For example, stopping crawling at $t_f$ saves approximately 50% cost compared to stopping at $t_i$.

Assume $W$ is an obtained data window with the size of $ws$, for a query $q_i$, if for $i < ws$, $W_i = (q_i \lor \ldots \lor q_i)$, if $i \geq ws$, $W_i = (q_{i-ws+1} \lor \ldots \lor q_i)$, $cost_{max}$ is the maximum available resource that a crawler has, and $e$ is a very small positive number, we present an strategy for stopping crawling: after submitting $q_i$, if $\sum_{i=1}^{\infty} cost(q_i) < cost_{max} \land W_i > e$ is false, then stop crawling at $q_i$.

The value of $ws$ depends on $cost_{max}$. The greater the value of $cost_{max}$, the bigger $ws$ can reach. When $ws$ is determined, $e$ becomes the critical factor in the strategy for stopping crawling. If $e$ is too small, the stopping time will be prolonged, while more data records may be obtained. To the contrary, the crawler will stop at an earlier time and reduce quantity of records retrieved. Therefore, the value of $e$ depends on the importance of the resource cost and the crawling data.

5 EXPERIMENTS

To evaluate the performance of our MEP-based adaptive query method, we performed experiments on 6 real Deep Web sites and compared our method with the representative methods of both non-prior knowledge and prior knowledge-based methods, i.e. adaptive query method based on single IDE (Infinite Domain Element) (Ntoulas, 2005) and classical LVS method (Raghavan, 2001).

5.1 Effectiveness of MEP-based Adaptive Query

We test our MEP adaptive query method on 6 real Deep Web sites, and the detailed experimental results are shown in Table 1. Results show that our method is quite effective for crawling Deep Web.

<table>
<thead>
<tr>
<th>URL (http)</th>
<th>Domain</th>
<th>Size/ Harvest/query NO.</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.jos.org.cn">www.jos.org.cn</a></td>
<td>Paper</td>
<td>1380/1380/143</td>
</tr>
<tr>
<td>ejc.ict.ac.cn</td>
<td>Paper</td>
<td>2523/2523/13</td>
</tr>
<tr>
<td><a href="http://www.jdxb.cn">www.jdxb.cn</a></td>
<td>Paper</td>
<td>424/424/16</td>
</tr>
<tr>
<td><a href="http://www.paperopen.com">www.paperopen.com</a></td>
<td>Paper</td>
<td>743,444/730,000/39</td>
</tr>
<tr>
<td>vod.xjtu.edu.cn</td>
<td>Movie</td>
<td>700/679/311</td>
</tr>
<tr>
<td>music.xjtu.edu.cn</td>
<td>Music</td>
<td>154,000/146,967/386</td>
</tr>
</tbody>
</table>
5.2 Performance Comparison

We compare the crawling performance of our MEP-based adaptive query (MEP or MEP adaptive for short) method with the adaptive query method in (Ntoulas, 2005). The experimental result is shown in Fig. 6, where the x-axis represents the query numbers and y-axis plots the database coverage.

We find that if the query form contains a FMEP, our method shows distinct advantage over the adaptive query method based on single IDE (see Table 1). In order to evaluate performance of our method on query forms only containing IMEPs, we conduct other two experiments on Babe Raccoon and Blue Lotus. Fig. 6(a) shows the experimental results on Babe Raccoon, in which IDE1, IDE2, IDE3 represent the crawling curve of the method in (Ntoulas, 2005) on elements “movie name”, “actor”, “director” respectively and MEP denotes the curve of our method on three IMEPs. The experimental results on Blue Lotus are plotted in Fig. 6(b), in which IDE1, IDE2, IDE3 represent the crawling curve of the method in (Ntoulas, 2005) on element “author”, “publisher”, “title” respectively and MEP denotes the curve of our method on these IMEPs. Fig. 6(a) and 6(b) indicate that our method is more efficient than the adaptive query method in (Ntoulas, 2005) on query forms only containing IMEPs.

The MEP adaptive method is based on multi-pattern, and usually there are several MEPs to be selected for each query. If each query takes the same MEP, the method degenerates into single IDE-based. From this point of view, the adaptive query method based on single infinite element in (Ntoulas, 2005) can be regarded as a special case of the MEP-based adaptive method. Single pattern always brings locality of the candidate keywords, while multi-pattern can make full use of each pattern’s query capacity, and break through the locality of the keywords selection which leads to the problem of “data islands”.

In order to compare the crawling performance between the MEP adaptive method and the prior knowledge-based methods, we conduct experiments on the site of Blue Lotus. In addition, we find that the keyword with high value of \( P(q_i) \) in our method is useful for the LVS method. If we update such keyword’s \( Mv \) value with its corresponding \( P(q_i) \) in the LVS table and then perform the LVS method (Raghavan, 2001), it can remarkably improve crawling efficiency. We call it the enhanced LVS method. Fig. 7 shows the experimental results.

According to Fig. 7, we can gain the efficiency order of the three methods: the MEP adaptive method > enhanced LVS method > classical LVS method. Since the enhanced LVS method uses the updated LVS table which improves the veracity of the prior knowledge in LVS table, it contributes to better crawling efficiency. However, because of the inherent deficiencies of the prior knowledge-based methods, it is still less efficient than the MEP adaptive method.
6 CONCLUSIONS

Due to the heterogeneity among the query forms, effective Deep Web crawling is a difficult issue. In this paper, we propose the concept of MEP and a MEP-based Deep Web adaptive query method. The proposed method, like the prior knowledge-based methods, has the ability to handle a variety of web forms. Moreover, the method has an advantage of good crawling efficiency over the non-prior knowledge methods, and can overcome the problem of “data islands” to a certain extent. Performance comparisons with the related method validate the better query capability of our method.

Although our method can be effective for most Deep Web sites, it has the following limitations: first, it may not perform well for some Deep Web sites that limit the size of the result set; second, it cannot support Boolean logic operators (e.g. AND, OR, NOT) in queries. We will focus on these issues in our future work.

ACKNOWLEDGEMENTS

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. 2006BAK11B02, 2006BAJ07B06.

REFERENCES


