Retrieving Deep Web Data Based on Heuristic Hierarchy Tree Model

Xuehuan Li$^{1,2}$, Zhiming Cui$^{1}$, Pengpeng Zhao$^{1,*}$, Jiumei Mao$^{1,2}$

$^1$Department of Computer Science and Technology, Soochow University, Suzhou 215006, China
$^2$Production Information Technology Co., Ltd of Suzhou, Suzhou 215021, China

Abstract

Deep Web data refers to a dataset that allows user to query through a search interface, and be rendered in dynamically generated web page, generally topic-based. However, many web database interfaces limit the number k of relevant tuples returned for each query submitted by user, which denotes top-k problem. To address this problem, we propose a novel method to prune hierarchy tree, which aims at solving web database tuples siphoning problem which interface has multi-attributes interface and top-k problem. The method transforms data retrieval problem into a hierarchy tree construction problem while uses heuristic information to prune the tree. Experiments over controllable databases and real Deep Web site confirm the efficiency of our approach.

Keywords: Deep Web Data; Top-k; Heuristic Hierarchy Tree Model; Multi-attributes Interface; Data Retrieval

1 Introduction

Today, many web database interfaces limit the number k of relevant tuples returned for each query submitted by user according to scoring function, and if we submit the repeating same query the crawler may not retrieve new tuples, which we call top-k restriction problem. For instance, interface of taobao.com with restriction of 100 pages per query, which limit number of pages turns, and a maximum of 3000 records returned for per query in interface of nsf.gov/awarsearch, single query is unable to get all tuples which satisfying the query condition of the target web database. Top-k problem is a challenge for us to achieve data extraction task, and it limits many valuable applications, such as the value comparison in shopping of electronic commerce, vertical search, data mining and data analysis and so on.

*Project supported by the National Nature Science Foundation of China (No. 61003054, No. 61170020), College Natural Science Research project of Jiangsu Province (No. 10KJB520018), Science and Technology Support Program of Suzhou (No. SG201257), Science and Technology Support program of Jiangsu province (No. BE2012075).

*Corresponding author.
Email address: ppzhao@suda.edu.cn (Pengpeng Zhao).
Extensive previous research has carried out on Deep Web data extraction task that based
simple search interface. A web crawler was proposed in [1] extract tuples from keyword based
Xiaofeng etc. [3] put forward a web database sampling method based on graph model, which
use sample dataset which are stored in the local to generate the next query, to get data source
approximation of random samples. Wang Yan [4] etc. converts the extraction problem into a set
covering problem by a weighted greedy algorithm. A query selection strategy is presented in [5]
based on the graph model of attribute value. In paper [6], gustavo etc. proposed an automated
query keyword selection of text domain for keyword interface.

All researches mentioned above are appropriate for data extraction of web databases that
provide simple query interface and have not the top-k problem, but the extraction efficiency of
multiple attribute interface is not very well. To solve this problem, Zhang Zhuo etc. [7] introduced
the formal concept to Web data extraction, and then Jianwei Tian etc. [8] established a query
expansion method based on hierarchical tree model. Sheng, etc. [9] proposed a query
method according to the attribute types. In this paper, we propose a novel algorithm based on hierarchy
tree model and attribute types proposed in [8,9].

The problem we want to solve is how to get most of non-repetitive data records, while min-
imizing the number of queries to send. The paper is organized as follows: the second section
gives a description of practical problem we will face, the third section we establish a heuristic
hierarchical tree model and analyze cost of the model, in the fourth section extensive experiments
are performed on real datasets verifies the efficiency of the proposed algorithm, at last section,
we give a summary and outlook.

2 Problem Description

Fig. 1 shows the form search interface of Yahoo Autos. We can divide entity attribute into
different types depend on whether an attribute has a total order: 1) Numeric attribute, such as
price, mileage and so on; 2) Categorical attribute, such as make, Model, and Body Style, etc.
The range interval of numeric attribute can be treated as Categorical filter attribute.

In our problem description, the crawler need know the domain of each attribute in the form,
but how to obtain domain of attributes is not our main research topic, domain discovery has been
studied in [10], we can use the algorithms proposed in it to get attribute domain. We have reason
to assume that: 1) domain of each attribute is known, it can be obtained from the background
knowledge or the drop-down list of query form; 2) if the interface does not provide refined query
of numeric attributes, we utilize fixed value interval of numeric as a value of categorical attribute.
This paper is to retrieve the entire dataset of hidden database which provides multi-attribute
query interface and has top-k problem while minimizing the number of queries.

3 Model of Hidden Databases

- **Data space D.** Suppose the entity has \( d \) attributes \( A_1, A_2, \ldots, A_d \), and each of which has
  no disjoint domain. Then data space is the Cartesian product of \( \text{dom}(A_1) \times \text{dom}(A_2) \times \cdots \times \text{dom}(A_d) \), but not every point in data space has corresponding tuples in the real dataset.
We refer to the number of domain of attribute $A_i$ as $N_i$.

- **Query answer.** That is a subset of $q(D)$, the bag of tuples in D which match $q$: If $|q(D)|>k$, an overflow occurs and only the top-k results are returned, along an overflow flag indicating that more tuples matching the query cannot be returned. If $|q(D)|=0$, then an underflow occurs as no tuples match the query. Otherwise, i.e., when $|q(D)| \in [1, k]$, we say that $q$ is valid. $|q(D)| \leq [1, k]$, which is referred to as resolved query, otherwise it is overflow.

- **Single-value query.** We call a query $q$ is a SQ if its predicates have the from: $A_1 = \ast, \ldots, A_{i-1} = \ast, A_i = c, A_{i+1} = \ast, \ldots, A_d = \ast$, where $c$ is a value in $dom(A_i)$, which short for SQ. Namely, the query has a wildcard predicate on all but one attribute $A_i$ for some $i \in [1, d]$. We use the notation $A_i = c$ to uniquely refer to a SQ. Clearly, varying $c$ in $dom(A_i)$ defines $N_i$ slice queries, such that the total number of slice queries of all dimensions is $\sum_{i=1}^{k} N_i$.

- **Heuristic-DFS.** We record response of SQ $q$ in a lookup table, which stores the response of query ($q$). If the $|q(D)| \leq [1, k]$, then we record the response tuples in table, whereas if $|q(D)| > k$, we record a overflow signal. In the process of constructing query tree, we refer to the method that adopting depth-first extension method while using the lookup table maintained in local to pruning the tree as heuristic-DFS.

## 4 Heuristic Hierarchical Tree Model

### 4.1 Modeling of the tree

Data extraction problem in Web database can be converted into the problem of traversal tree based on the hierarchical tree model, we can extract the tuples through traversal hierarchy tree. The construction of heuristic hierarchical tree model is as follows: Firstly, we submit SQ $A_1 = c$, $c \in dom(A_1)$, and record the result in the locally lookup table. Then for each node $u$, if its corresponding query($u$) is overflow, we call function extended-DFS with node $u$ as input parameter, it returns all tuples that satisfy expansion subtree of node $u$.

In function extended-DFS, input parameter node $u$ in level $l (l \in [0, d-1])$ as root of subtree expanded from node $u$, for each child node $v$ of $u$, we select randomly value $c$ of attribute $A_{i+1}$, $c \in dom(A_{i+1})$ in level $l+1$, and add it to the query path to make a new query($v$). Firstly, we can
search the lookup table, check whether result of $\text{query}(v)$ is completely obtained from result of SQ $A_{t+1}=c$, if result of SQ $A_{t+1}=c$ is not overflow, then we can obtain the satisfying results from local lookup table, in which case the subtree of $v$ does not need to be built, otherwise, forward by sending $\text{query}(v)$ to server, the response of this query results in two cases:

- Response is overflow, then the above process will be extended recursively, until extend to the highest dimension $d$ or find a resolved node;
- Otherwise, we find a resolved node, to extract and store the response. Then, process backtrack to parent node, continues to traverse the brother nodes by constructing query which using the other attribute values of parent node.

Obviously that the whole process is regarded as a process to build a query tree based on depth-first method, while using the lookup table maintained in local to pruning the extension tree effectively.

**Example 1.** We use an example to explain the establishing of the tree model. The data distribution as shown in Fig. 2(a). A sample dataset of 12 tuples in the whole data space, $k$ is set to be 3, there are 4 distinct values in domain of each attribute. Firstly, node $U_1$ is the root of tree model, as shown in Fig. 2(a), we know that its corresponding $\text{query}(U_1)$ is $A_i = *(1 \leq i \leq d)$, so this query can be ignored, because the response must be overflow according to prior knowledge.
Then for $\forall x \in \text{dom}(A_1)$, we perform the SQ $A_1 = x$, and record the response in the first row of local lookup table, which is shown in Table 1.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1$</td>
<td>Ovflow</td>
<td>${t_4, t_5}$</td>
<td>${t_6, t_7}$</td>
<td>Ovflow</td>
</tr>
<tr>
<td>$A_2$</td>
<td>${t_1, t_6, t_8}$</td>
<td>Ovflow</td>
<td>${t_3, t_5, t_{11}}$</td>
<td>${t_{12}}$</td>
</tr>
</tbody>
</table>

### 4.2 Analysis of the model cost

We assume that the query interface has limitation of size $k$, the target database has $n$ tuples to be retrieved, the data space of which has $d$ dimensions, then we analyze cost of this hierarchy tree model constructed by using heuristic-DFS algorithm, let $T$ as the subtree of the model tree, each node in it should be extended recursively.

As shown in Fig. 2(b), the black node in tree represent the query that need to be sent, namely internal node, which should be extended recursively. The concentric circles node denotes that the query can get result from the lookup table, it is unnecessary to be sent to the server, so we can eliminate it from $T$. We define the cost of the model as the sum of number of SQ plus the numbers of nodes in $T$. For example, $T$ of example 1 includes nodes $U_1$, $U_2$ and $U_5$, $U_7$, $U_{11}$ shown in Fig. 2b.

**Lemma 1** When $d = 1$, the number of queries is $N_1$;

**Lemma 2** When $d > 1$, the most cost is $\sum_{i=1}^{d} N_i + \frac{n}{k} \cdot \sum_{i=1}^{d} \min(\frac{n}{k}, N_i)$;

**Proof** when $d = 1$, the number of distinct values of domain $A_1$ is $N_1$, so number of queries we need to send is $N_1$, the process ends.

When $d > 1$, we consider the worst case, that is to say the SQ of all $d$ dimensions should be performed. Next we will proof that the internal nodes in the tree $T$ is at most $\frac{n}{k} \cdot \sum_{i=1}^{d} \min(\frac{n}{k}, N_i)$.

For each level of extended tree, $i \in [1, d]$, the set of internal nodes in the $i$-th level in $T$ is denoted by $S_i$, refers the set of overflow nodes in $i$-th level. Next step, we proof the number of nodes in $i$-th level is at most $\frac{n}{k} \cdot \sum_{i=1}^{d} \min(\frac{n}{k}, N_i)$. We only need to proof the numbers of the internal nodes in the level of $i = 1$ is at most $\frac{n}{k}$, and the number of children of each internal nodes is at most $\min(\frac{n}{k}, N_i)$.

Let an internal node in the level of $i = 1$ is $u$, denotes $n_u$ as the number of tuples that satisfying $\text{query}(u)$ in target data space $D$, it must be $n_u > k$, otherwise its subtree will be pruned, which is a contradiction that $u$ is an internal node. And together with $\sum_{u \in S_{i-1}} n_u < n$, we conclude that the number of internal nodes in $S_{i-1}$ is not more than $\frac{n}{k}$. If it is extended to the $i$-th level, for each node $v \in \text{children}(u)$ refines $\text{query}(u)$, by replacing $A_i = \ast$ to $A_i = c$. If the SQ $A_i = c$ is not overflow, the query can be answered locally, the node $v$ cannot appear in subtree of $u$, otherwise, if $n_v > k$, it forms a node in level $i$ of $T$. For query results in the same level have no intersection, so $\sum_{v \in \text{dom}(A_i)} n_v \leq n$, so the number of internal nodes in the subtree of $u$ is smaller than $\frac{n}{k}$. Obviously it is no more than $N_i$, because the internal nodes of the $i$-th
level cannot more than the number of SQ in the \( i \)-th dimension, namely \( N_i \). To sum up, the number of internal nodes in the \( i \)-th level \( |S_i| \leq \min\left(\frac{n}{k}, N_i\right) \), and the depth of \( T \) is at most \( d \), so 
\[
\max_{\text{cost}} < \sum_{i=1}^{d} N_i + \frac{n}{k} \cdot \sum_{i=1}^{d} \min\left(\frac{n}{k}, N_i\right).
\]

### 4.3 The Heuristic-DFS algorithm

First we order attributes according to size of attribute domain, based on the theory that be proved in [8], namely \( |\text{dom}(A_1)| \leq |\text{dom}(A_2)| \leq \cdots \leq |\text{dom}(A_m)| \), then we can obtain the minimum query space. The returned data are maintained in a local database \( DB_{\text{local}} \), which is initialized by \( \phi \). The Heuristic-DFS algorithm is described below:

**Step 1.** Firstly, we perform SQ of 1st dimension, then for each SQ that is overflow we call function \emph{extended-DFS} to refine the query condition, the process of the extension is recursively.

**Step 2.** In function \emph{extended-DFS}, firstly, if it has reached the highest dimension \( d \) of data space, then it come to an end. Otherwise, if it is the first time the algorithm extends to the current dimension, then it calls function \emph{record-SQ} to execute SQ of current dimension, record response in lookup table, and then go to step 3.

**Step 3.** We look for the lookup table to see whether the corresponding SQ is overflow or not, if it is not overflow, we can find the tuples that satisfying the extended condition from local table, and then backtrack to parent node, recurs in depth-first manner. Otherwise, we need to construct and submit the query, the response results in two cases: 1) overflow, then call function \emph{extended-DFS} to extend the query recursively; 2) otherwise, the returned results are merged together with \( DB_{\text{local}} \).

### 5 Experiments

#### 5.1 Datasets of experiment

In this paper, we perform experiments on controlled real datasets, and construct local database server with real datasets, so we can control the size of \( k \). we sort the properties according to the numbers of the value domain from small to large according to theory proposed in [8].

- The first dataset. The common test dataset loaded from \textit{archive.ics.uci.edu/ml/datasets/Adult}, there are 48842 data records contain 6 categorical type attributes \{sex, race, relationship, marital-status, work Class, education\}.

- The second dataset. About 120000 more data records about shoes have been fetched from Taobao, it has 7 attributes of categorical types \{season, style, popular, close, toe, heel, material\}.

- The third dataset. Common test dataset from CarEvaluation which has 6 categorical attributes \{buying, maint, doors, persons, lug-boot, safety\}, loaded from \textit{archive.ics.uci.edu/ml/datasets/CarEvaluation}. 


5.2 Analysis of experimental results

In Fig. 3, it shows the cost comparison between Heuristic-DFS algorithm and basic DFS algorithm on three controllable datasets and a real Deep web site of Yahoo Auto under the limitation of different k, the cost of the former shows lower cost than the latter.

As we know, our method can get as same non-repetitive data records as tree model proposed in [8] at last, but the number of submitting query is less in our model, so the method proposed in this paper is better than basic DFS, and it verifies the validity of the algorithm. The result has its rationality, the degree distribution of the attribute value graph is very close to power-law distribution, i.e. a few attribute values are extremely popular, while “the massive many” are sparsely connected, it approximately obeys the exponential distribution law, and the proposed method can be effectively carried out on the hierarchy tree pruning to reduce the query cost.

6 Summary and Outlook

Based on the predecessors’ research, in this paper, we propose a new method to prune the hierarchical tree while using smaller number of queries to get non-repetitive tuples from target database. To solve the top-k problem with comprehensive interface, we also need solve keyword
selection problem of text domain, the next step is to study query strategy which takes categorical type combine with keyword selection problem of text domain. The goal is to break through the top-k barrier of the comprehensive form interface, and explore an extraction method to retrieve entity $D$ in hidden database while minimizing the number of queries.

References


