XPath-Wrapper Induction for Data Extraction

Nam-Khanh Tran
college of Engineering and Technology
Vietnam National University, Hanoi
khanhtr09@vnu.edu.vn

Kim-Cuong Pham
Computer science department, University
of Illinois at Urbana-Champaign
kincuong@gmail.com

Quang-Thuy Ha
College of Engineering and Technology
Vietnam National University, Hanoi
thuyhq@vnu.edu.vn

Abstract - The Web contains an enormous amount of information which is formatted for human beings. This makes it difficult for computer to extract relevant content from various sources. This paper presents an XPath-wrapper induction algorithm which leverages user queries and template-based sites for extracting structured information. Our experiments show average accuracy of 94%.

I. INTRODUCTION

A large amount of information on the Web is presented in regularly structured objects. Each object is often presented by detail page containing attributes about that object. These pages are dynamically generated by populating fixed page templates with content from a back-end DBMS. Previous study [1] finds that 40-50% of web content consists of webpage templates and that is growing approximately 6-8% per year. As an example, consider the detail page shown in Fig. 1 for laptop “Acer Aspire” from ecomelelectronics.com. The page contains a wealth of information including details like laptop name, price, hard drive capacity, processor speed, etc. Extracting attributes is useful because it allows us to integrate information from multiple sources to create extensive database of entities. The objective of our work is to automatically create wrappers that extract object attribute’s values from detail pages.

Web wrappers are popular tools for efficiently extracting information from web pages. In recent years, a number of research papers in this area have studied [2][3][4][6]. Even though it is possible to learn correct wrappers, most of methods required human-annotated samples. Hence, some recent research has been minimizing the number of examples the users have to annotate [8]. Still, even though user effort is significantly reduced, the amount and rate of information growth on the Web will rapidly overwhelm human effort. There also exist several other automatic approaches, Exalg [6] and RoadRunner [2] which can automatically extract data from data-intensive web sites. Either Exalg or RoadRunner use structured information of sample pages to create HTML tags-based templates. A template contains two parts: constant parts of the HTML sources’ string sequence and variant part viewed as the right data to be extracted.

However, previous wrapper induction algorithms have not leveraged user knowledge, e.g., user queries into learning wrappers. We observe that this is useful because search result returned for such user queries often contain relevant detail pages. Moreover, they also contain different pages from the same template-based sites. Queries are usually ambiguous, short, keyword-based approximations of often-underspecified user information needs. An intriguing aspect of queries, however, is the ability of indirectly capture user knowledge, precisely as they inquire about what is already known. Therefore, user queries are not only used to request new information, but also indirectly convey knowledge in the process. Pasca et al. [5] is the first to study the effect of user queries on extracting prominent attributes of classes (e.g., top speed, price and fuel consumption for CarModel) from unstructured text. Pasca et al. [5], however, treated queries and documents separately as self-sufficient data sources even though they actually have close relation on extracting attribute values.

In this paper, we proposed an XPath-wrapper induction algorithm that leverages user knowledge, e.g., user queries and template-based websites to automatically extract wrappers and extract object attribute values. The attribute values allow us to create extensive database of extracted entities. The database can then be queried by users to access and compare object features between different sources.

Our contributions are as follows:

- We propose a new wrapper induction algorithm to automatically extract object attribute values, which does not require labor-intensive, training data. The method exploits the relationship between user queries and relevant result returned by search engines.
- We are the first to reduce the extraction problem as ranking XPath and propose an effective ranking function.

II. OVERVIEW OF THE PROBLEM

In this section, we give an overview of the problem of data extraction based on template-based web sites and user queries.
to aid in extraction. Formally, considering a user query \( Q \) and a set of query-relevant web pages \( D = \{ d_1, d_2, ..., d_k \} \) belong to the website \( W \) which contains a list of object attributes \( A = \{ a_1, a_2, ..., a_m \} \); in this paper, we address the problem of wrapper induction for each attribute from query \( Q \) and web pages \( D \).

A. User Query

In recent years, many approaches have studied to support structured query on web data [7][9]. Formally, a structured query is defined as a tuple of \( n \) constraints on object attributes, \( q = c_1 \land c_2 \land ... \land c_n \) where \( c_i \) is a constraint on attribute \( a_i \). For example, users want to search “a laptop Hp with capacity from 100Gb to 300Gb at most $1000”, a structured query can be “\( name = hp \land 100 \leq hdd \leq 300 \land price \leq 1000 \)”. In addition, search engines, e.g., Google, Yahoo, also allow users to pose structured-oriented queries such as “canon 5..10MP”. Users then obtain relevant-query documents with possible precision returned from search engines. In this paper, we leverage the structured queries to generate XPath-wrapper for object attribute.

B. Page Templates

Gibson et al. [1] find that 40-50% of Web content consists of webpage templates and that templatization is growing approximately 6-8% per year. The page template of a list page contains data that is shared by all list pages and is invariant from page to page. A different page template is used to generate detail pages.

Template-based sites have common properties. For example, pages within a website have a similar structure, thus attribute values occur at fixed positions within pages. The fix positions can be defined as paths from the root to the node called XPath containing attribute values on DOM tree of the pages. Once we have identified attribute values in a few pages of a site, we can infer their positions, and then use these to extract attribute values from remaining pages of the site. However, the task of extracting attribute values from these sites remains challenging because pages within site still have a lot of spurious information such as advertisement information or dynamic user generated comments, etc.

C. Problem

The very first step of extracting data from a website is that we identify their position in a few pages to infer their XPath. Then we use these XPath to extract attribute values from remaining pages of the site in the second step. In this paper, we focus on the first step, in literature called XPath-wrapper induction. Formally, the problem is defined as the following: Given a query and a set of query-relevant web pages \( D \) belong to a site \( W \), generate XPath-wrapper for an attribute. In this paper, we restrict our attention on detail web pages.

III. XPath-Wrapper Feature Generation

As mentioned earlier, our approach leverages user queries and query-relevant structured pages to generate XPath-wrappers and extract attribute values from websites. In this section, we first formally define key concepts used in the paper. We then introduce our approach to generate XPath-wrapper for object attribute.

Weighted XPath. An XPath statement (or path, synonymously) defines traversal through a DOM tree. It contains HTML attribute names and positions information. It is likely a path from root (tag <html>) to the tag that contains object attribute value. We observe that even though the templates of websites can vary largely, attribute values often occur under attractive formats (e.g., large font size, bold face) which allow users easily to recognize. Thus, we assign a weight for XPath to utilize such format information for extraction.

The weight of an XPath is sum of the weights of HTML tags in the XPath. In this paper, we utilize default weight assignments in TextNet32 [10]. Such weights found on the importance of HTML tags.

\[
\text{Weight}_{i} = \sum_{j=1}^{m} \text{Weight}_{T_{tag}}
\]

In which, \( m \) is the number of HTML tags in XPath \( X_i \). For example, for an XPath \( X_i = \langle /html/body/table/tr[3]/td[2]/b/text() \rangle \), \( X_i \) is assigned the weight of 2.0 which is sum of the HTML tag weights (html, body, table, tr, td, b), the weight of \(<b>\) tag is 2.0 and those of remaining tags are 0. When an XPath is assigned a high weight, its containing node value can be highly attribute value we need to extract.

Frequency of XPath. For an XPath \( X_i \), let \( F(X_i) \) denote the frequency of \( X_i \); that is, the number of pages within a website which contain \( X_i \). We call \( L_j \) is the list of possible XPath in web page \( d \) and a function \( g(X_i, L_j) = \begin{cases} 1 & \text{if } X_i \in L_j \\ 0 & \text{otherwise} \end{cases} \). Then, \( F(X_i) = \sum_{j=1}^{k} g(X_i, L_j) \) in which \( k \) is the number of web pages within the website. For template-based sites, attribute values often occur at fixed positions in web pages within sites; thus, when an XPath occurs in lots of pages, the XPath can highly contain attribute value.

Inverse Frequency of XPath. For an XPath \( X_i \), let \( IF(X_i, C) \) denote the inverse frequency of \( X_i \) with constraint \( C \); that is, the number of pages within a website which by using \( X_i \) can extract attribute values from those satisfied constraint \( C \). The constraint \( C \) depends on what attribute we learn wrapper; for example, the value of \( price \) attribute must be number, \( capacity \) attribute must be number and \( brand \) attribute must be string, etc. When we determine XPath from webpage \( d \) and user query (e.g. “\( price \leq 1000 \)”), the XPath cannot be learned from other web pages which have \( price > 1000 \) because web pages returned from search engines are not perfectly relevant to the query. So, inverse frequency infers the ability of XPath to extract correct values from web pages it is not learned.

IV. XPath-Wrapper Algorithm

Our algorithm proceeds as following steps:

- **XPath determination:** from the query \( q \) and a web page \( d \) within site \( W \), we determine a list of possible XPath. Through a list of web pages within web site \( W \), we have a set of possible XPath lists \( L = \{ L_1, L_2, ..., L_k \} \).
- **XPath elimination**: from all possible XPath candidate \( L_1 \), we eliminate those that are repetitive because that indicates unimportant information within a detail page.
- **Feature calculation**: from eliminated XPath lists \( L \), we calculate feature values (weight, frequency, inverse frequency) for each XPath in \( L \).
- **XPath ranking**: rank XPath candidates with the features.

The following sections describe these steps in detail.

### A. XPath Determination

From the query \( q \) and web page \( d \), we need to determine a list of possible XPath. We firstly parse the web page \( d \) to DOM tree and then with the aid of the query, we find all leaf nodes containing the values satisfied the query; as a result, a list of paths from root (tag `<html>`) to the leaf nodes (XPath) is determined.

For example, the query user posed is “*sony vaio*, price \( \leq 1000 \)” and the DOM tree of the webpage \( d \) is presented in figure 3.

If a leaf node, which is denoted `#text`, contains a string “*sony vaio*” or a number “*number \( \leq 1000 \)*”, the path from root to the leaf node is determined as XPath candidates.

### B. XPath Elimination

In web pages, there is a lot of noisy information such as advertisement. Such information usually appear on most of web pages within a website includes similar information (similar objects), special products, the most popular products, etc. The paths from root to the node containing the spurious information have significantly high frequency; thus, eliminating such information helps to improve the precision of learning wrapper process.

With all possible XPath candidates in \( L_1 \), we eliminate sets of XPath that have similar structures. Two XPath have similar structures if they are the same when we ignore the index of HTML node. For example, XPath `/html/body/table[1]/tbody/tr[2]/td[3]/text()` is similar to XPath `/html/body/table[1]/tbody/tr[2]/td[4]/text()` and XPath `/html/body/table[1]/tbody/tr[2]/td[6]/text()`.

### C. XPath-Feature Calculation

An XPath includes three features: weight, frequency, inverse frequency. These features are used to rank XPath in order to learn the XPath-wrapper.

We compute such XPath features as described in previous section.

### D. XPath Ranking

XPath ranking is based on the visual emphasis of HTML elements that the attribute values are explicitly denoted in a special way in order to get attention from users. Otherwise, the XPath containing attribute value trend to have high weight; as a result, the higher weighted XPath is the higher rank it has.

In addition, because attribute values often occur at fixed positions, the XPath containing these values are determined in most of web pages. Thus, we infer that the higher frequency XPath gets, the higher rank it has. Moreover, the inverse frequency refers the ability of XPath to extract correct value from web pages which it is not determined. Like the frequency of XPath, when an XPath has higher inverse frequency, it has higher rank.

Based on these observations, we proposed a ranking function with constants \( \alpha, \beta \in [0,1] \) to rank featured-XPath candidates.

\[
Score(X_i) = \alpha \cdot F(X_i) + \beta \cdot IF(X_i) + (1 - \alpha - \beta) \cdot Weight(X_i) \tag{1}
\]

Through experiments we determined that when \( \alpha = 0.5, \beta = 0.3 \) we obtains the highest accuracy.

### V. Experiments

We have performed experiments on two application domain camera and laptop. For camera domain, we have taken one query, 2890 related-query documents from 323 different sites and applied our approach for PRICE attribute. For laptop domain, we have built an object-oriented search engine following the approach described in [7] on 10574 documents.
We then posed five queries to the search engine and applied our approach on 408 returned documents from 20 sites for NAME, PRICE, PROCESSOR, CAPACITY attribute.

The queries we have taken are popular queries from some different users.

<table>
<thead>
<tr>
<th>TABLE I. LIST OF QUERIES</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Camera domain</strong></td>
</tr>
<tr>
<td>Query 1: brand name: canon, price: $200..1000</td>
</tr>
<tr>
<td>Query 2: brand name: sony viao, capacity: 100..200gb, processor: 1..3ghz, price: $1000..5000</td>
</tr>
<tr>
<td><strong>Laptop domain</strong></td>
</tr>
<tr>
<td>Query 1: brand name: dell, capacity: 80..300gb, processor: 1.5..3.5ghz, price: $1000..2500</td>
</tr>
<tr>
<td>Query 2: brand name: hp, capacity: 300..400gb, processor: 2..4ghz, price: $5000..3000</td>
</tr>
<tr>
<td>Query 3: brand name: acer, capacity: 80..250gb, processor: 2..4ghz, price: $200..2500</td>
</tr>
<tr>
<td>Query 4: brand name: lenovo, capacity: 100..400gb, processor: 1..4ghz, price: $5000..3000</td>
</tr>
</tbody>
</table>

Moreover, we have experimentally found the optional values for the most efficient extraction in the formula (1) $\alpha = 0.5$, $\beta = 0.3$.

<table>
<thead>
<tr>
<th>TABLE II. PRECISION OF XPATH-WRAPPER GENERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Price</td>
</tr>
<tr>
<td>Camera</td>
</tr>
<tr>
<td>Laptop</td>
</tr>
</tbody>
</table>

Table II presents the promising precision of our approach on two domains camera and laptop.

VI. RELATED WORKS

Wrapper induction has attracted a considerable amount of attention from research community [2][3][4][6]. However, to the best of our knowledge, none of the existing wrapper induction research leverages user queries to rank XPath in order to extract structured content. Early web information extraction methods were based on wrapper-induction [3][4]. Essentially, wrappers are learned from a few human-annotated sample pages; consequently, wrapper induction can be labor-intensive and expensive. To bypass this limitation, recent research has tried to minimize the number of examples users have to label [8]. Still, even though user effort is significantly reduced, the amount and rate of growth information on the Web will rapidly overwhelm human effort. There also exist several other automatic approaches, Exalg [6] and RoadRunner [2] which can automatically extract data from data-intensive web sites. A drawback of these approaches is that they work on the sequential HTML tag structure thereby missing the opportunity to exploit the underlying DOM structure which provides richer descriptive power for the wrappers. In addition, the approaches did not care about user queries which have considerable effect on data extraction [5]. M. Pasca et al. [5] has studied to use user queries on extracting prominent attributes of classes (e.g., top speed, price and fuel consumption for CarModel) from unstructured text. However, they treated queries and web pages separately as self-sufficient data sources even though queries and web pages actually have close relation on extracting attribute values. The origins of XPath wrappers can be traced to [11] but the wrappers in this work were hand-created while we reduce the problem of XPath wrapper learning to ranking problem by assigning features to XPath (weight, frequency, inverse frequency) which are done in an unsupervised way.

VII. CONCLUSION

As many template-based websites contain structured object information are growing, the requirement of extracting object attribute values also increase. In this paper, we presented a new wrapper induction algorithm which leverages user queries and template-based sites to automatically extract attribute values. Our experiments show promising result with average precision of 94% and the ability to adapt our algorithm into different domain for various object attributes.

Acknowledgement

This work was partly supported by the research project VNU QG.10.38 Research on model and solution development for Vietnamese Entity Search Engine and implementation.

REFERENCES