Automatic Wrapper Generation for Search Engines
Based on Visual Representation
G.V.Subba Rao, K.Ramesh
Department of CS, KIET, Kakinada, JNTUK, A.P
Assistant Professor, KIET, JNTUK, A.P, India.
gvrs888@gmail.com

Abstract:
Normally web databases are provides the information based on query representation. All the results are displayed without any structure. All extracted results are not provides any specific or perfect generation. It can contain some number of unnecessary, noisy and irrelevant web pages of information. These web database are not contains any interfaces implementation and cannot provide any desirable results specification. Present System is going to display with minimization results at extraction phase by implementing some of the machine processable techniques. These machine processable events are provides inheritance based results identification. We are going to implement some of the interfaces and wrappers and provide desired, accurate results representation. It can contain two types of extraction phase. First phase web data record extraction phase and web data item extraction phase. Each and every phase identifies the results like visual block tree, alignment of nodes process. We are going to extract the information based on Label and language independent process.

Key words -noise, data extraction, interface, wrapper, visual block tree.

I. Introduction

The problem of Web data extraction has received a lot of attention in recent years and most of the proposed solutions are based on analyzing the HTML source code of Web pages. However, HTML itself is still evolving (from version 2.0 to the current version 4.01, and version 5.0 is being drafted) and when new versions or new tags are introduced, the previous works will have to be amended repeatedly to adapt to new versions or new tags. In order to make Web pages vivid and colorful, Web page designers are using more and more complex JavaScript and CSS. Based on our observation from a large number of real Web pages, especially deep Web pages, the underlying structure of current Web pages is more complicated than ever and is far different from their layouts on Web browsers. This makes it more difficult for existing solutions to infer the regularity of the structure of Web pages by only analyzing the tag structures.

There are many search engines on the Web (e.g., [1, 2], where Web users not only interact with search engines, also many Web applications need to interact with search engines. For example, meta search engines utilize existing search engines to perform search [4] and need to extract the search results from the result pages returned by the search engines used. As another example, deep web crawling is to crawl documents or data records from (deep web) search engines [6] and it too needs to extract the search results from the result pages returned by search engines.

As another example, our Web Scales project [3, 5] aims to connect to hundreds of thousands of search engines and it is not practical to manually construct a wrapper for each search engine. Therefore, we need an automated solution. For search engines that have a Web services interface like Google and Amazon.com, automated tools may be used to extract their SRRs because the result formats are clearly described in the WSDL file of the Web Services[9,10].
In this work, we explore the visual regularity of the data records and data items on deep Web pages and propose a novel vision-based approach, Vision-based Data Extractor (ViDE), to extract structured results from deep Web pages automatically. Our approach employs a four-step strategy. First, given a sample deep Web page from a Web database, obtain its visual representation and transform it into a Visual Block tree which will be introduced later; second, extract data records from the Visual Block tree; third, partition extracted data records into data items and align the data items of the same semantic together; and fourth, generate visual wrappers (a set of visual extraction rules) for the Web database based on sample deep Web pages such that both data record extraction and data item extraction for new deep Web pages that are from the same Web database can be carried out more efficiently using the visual wrappers.

II. System Architecture

Figure 1 shows the architecture of our automatic wrapper generation system. The input to the system is the URL of a search engine’s interface page, which contains an HTML form used to accept user queries. The output of the system is a wrapper for the search engine. The search engine form extractor figures out how to connect to the search engine using the information available in the HTML form. Based on the extracted form information, the query sender component sends queries to the search engine and receives result pages returned by the search engine [6].

• We describe a complete system for wrapper generation that includes an interactive interface, a powerful extraction language, and techniques for deriving and ranking extraction patterns. The system was designed in a top-down fashion, by first considering how a user should be able to interact with the system, and then deriving techniques that enable this interface.
• To implement the interface, we describe a framework based on active learning that uses an additional verification set of pages to increase the robustness of the generated wrapper without requiring the labeling of many additional examples.
• We propose the use of a category utility function [7] to rank candidate tuple sets, in order to further decrease user effort.
• We perform a detailed experimental evaluation on 14 web sites that shows that our system can capture very challenging extraction scenarios with only minimal user effort.

Fig-2 Proposed System Architecture of AWGS

III. Deep Web Page Representation

The visual information of Web pages, which has been introduced above, can be obtained through the programming interface provided by Web browsers (i.e., IE). In this paper, we employ the VIPS algorithm [8] to transform a deep Web page into a Visual Block tree and extract the visual information. A Visual Block tree is actually a segmentation of a Web page. The root block represents the whole page, and each block in the tree corresponds to a rectangular region on the web page.

A. Visual Features of Deep Web Pages

Web pages are used to publish information to users, similar to other kinds of media, such as newspaper and TV. The designers often associate different types
of information with distinct visual characteristics (such as font, position, etc.) to make the information on Web pages easy to understand. As a result, visual features are important for identifying special information on Web pages. Deep Web pages are special Web pages that contain data records retrieved from Web databases, and we hypothesize that there are some distinct visual features for data records and data items. Our observation based on a large number of deep Web pages is consistent with this hypothesis [11,12].

**Position features (PFs).** These features indicate the location of the data region on a deep Web page.
- PF1: Data regions are always centered horizontally.
- PF2: The size of the data region is usually large relative to the area size of the whole page.

Since the data records are the contents in focus on deep Web pages, Web page designers always have the region containing the data records centrally and conspicuously placed on pages to capture the user’s attention. By investigating a large number of deep Web pages, we found two interesting facts. First, data regions are always located in the middle section horizontally on deep Web pages.

Second, the size of a data region is usually large when there are enough data records in the data region. The actual size of a data region may change greatly because it is not only influenced by the number of data records retrieved, but also by what information is included in each data record.

**Layout features (LFs).** These features indicate how the data records in the data region are typically arranged.
- LF1: The data records are usually aligned flush left in the data region.
- LF2: All data records are adjoining.
- LF3: Adjoining data records do not overlap, and the space between any two adjoining records is the same.

**Appearance features (AFs).** These features capture the visual features within data records.
- AF1: Data records are very similar in their appearances, and the similarity includes the sizes of the images they contain and the fonts they use.
- AF2: The data items of the same semantic in different data records have similar presentations with respect to position, size (image data item), and font (text data item).
- AF3: The neighboring text data items of different semantics often (not always) use distinguishable fonts.

**Content feature (CF).** These features hint the regularity of the contents in data records.
- CF1: The first data item in each data record is always of a mandatory type.
- CF2: The presentation of data items in data records follows a fixed order.
- CF3: There are often some fixed static texts in data records, which are not from the underlying Web database.

**IV. Wrapper Generation Algorithm**
We now give details of the wrapper generation process. To do so, we first define some internal data structures and concepts used in the algorithm. A dom path object is used to represent each highlighted attribute of the training tuple in the system. In particular, a dom path object of an attribute identifies all nodes on the path from the root to the leaf node(s).

Given these dom path objects, we find the lowest common ancestor (lca) of a training tuple, i.e., of the text nodes highlighted in the training tuple. An LCA object is a data structure that stores (i) a set of lca nodes for tuples on the training web page and (ii) a set of extraction patterns that extract this set of lcas. The major steps of our algorithm are as follows:
1. Initializing the internal representations: Internal representations are created through preprocessing of the training and verification Web pages.
2. Creating dom path and LCA objects: Upon retrieval of the training tuple, dom path objects are created for each attribute, and multiple LCA objects are created as follows (if possible):
   - Using the attribute dom path objects, create the lca dom path object of the training tuple.
   - Using the lca dom path object, create an extraction pattern with only tag Name(N, T) predicates at each level. Execute this pattern and identify a set of lcas:
     - Starting from the nodes in the largest lca set, move towards the root and identify the ancestor nodes at each depth.
   - Starting from the root and moving down on the lca dom object, construct expressions for each depth, where an expression is a conjunction or disjunction of predicates. The goal is to obtain expressions that accept different sets of nodes at each depth. One obvious restriction is that the node in the lca dom path of the training tuple has to accepted.
   - Create patterns by concatenating the expressions defined for each depth. Execute these patterns, and group those that extract the same set of lca nodes.
Create and return LCA objects for each unique set, containing the corresponding set of extraction patterns.

(3) Creating patterns that extract tuple attributes: Given the largest lca set, we now create patterns leading from the lcas down to the attributes. The algorithm for this step is similar to the above, but a different set of predicates is used for the leaf nodes. In order to avoid creating very unlikely patterns, we apply some heuristic basic filtering rules. For example, use of neighbor predicates in an expression may be unrealistic in some cases: If a neighbor node is visually too far on the web page, or the content of it is just a single whitespace, then we do not consider such predicates for the patterns.

(4) Creating initial wrappers: For each wrapper, the attribute extraction patterns are obtained by concatenating the patterns created in (3) to the patterns stored in the LCA objects in (2). Then the patterns for extracting tuple regions are created as follows:
First, the lcas of the candidate tuples are checked to see whether they are unique. If so, the set of patterns that define tuple regions is equal to the lca patterns. Otherwise, there should be some content on the web pages (such as lines, images, breaks, or some text) to visually separate the tuples. These visual separators are then identified in an iterative manner to create the tuple extraction patterns; see [9] for details.

(5) Generating the tuple validation rules and new wrappers:
If a validation rule does not change the tuple set, then this rule is placed in the unconfirmed rules set of that wrapper. If a rule causes a different tuple set, then a new wrapper is replicated from that wrapper, and the rule is stored in the set of confirmed rules of the new wrapper. If this is the desired tuple set, then this means this rule must apply to future tuples.

(6) Combining the wrappers: The wrappers are analyzed, and any that generate the same tuple sets are combined.

(7) Ranking the tuple sets: This crucial step is discussed below.

(8) Getting confirmation from the user: If one of the tuple sets is confirmed, the system continues with the next step. If the user could not find the correct set, then she can now select the largest tuple set with only correct tuples, highlight an additional training tuple missing from this set, and go back to step (2).

(9) Testing the wrapper on the verification set: Once the correct tuple set is obtained with the interactions, the corresponding wrapper is tested on the web pages in the verification set to resolve any disagreements. In the case of a disagreement, we again use ranking of tuple sets as in step (7).

(10) Storing the wrapper: The final wrapper is stored in a file.

V. Experimental Evaluation
We have implemented an operational deep Web data extraction system for ViDE based on the techniques we proposed. Our experiments are done on a Pentium 4 1.9 GH, 512 MB PC. The experiment is done using Java programming with HTML-based approaches. The output results are given as follows.
VI. Conclusion:

With the flourish of the deep Web, users have a great opportunity to benefit from such abundant information in it. In general, the desired information is embedded in the deep Web pages in the form of data records returned by Web databases when they respond to users’ queries. Therefore, it is an important task to extract the structured data from the deep Web pages for later processing. We focused on the structured Web data extraction problem, including data record extraction and data item extraction. First, we surveyed previous works on Web data extraction and investigated their inherent limitations. Meanwhile, we found that the visual information of Web pages can help us implement Web data extraction. “Based on our observations of a large number of deep Web pages, we identified a set of interesting common visual features that are useful for deep Web data extraction. Based on these visual features, we proposed a novel vision-based approach to extract structured data from deep Web pages, which can avoid the limitations of previous works. The main trait of this vision-based approach is that it primarily utilizes the visual features of deep Web pages. Our approach consists of four primary steps: Visual Block tree building, data record extraction, data item extraction, and visual wrapper generation. Visual Block tree building is to build the Visual Block tree for a given sample deep page using the VIPS algorithm. With the Visual Block tree, data record extraction and data item extraction are carried out based on our proposed visual features”.

Reference


