Journal of AI and Data Mining
Vol 6, No 2, 2018, 399-407
DOI: 10.22044/JADM.2017.990

Data Extraction using Content-Based Handles

A. Pouramini*, S. Khaje Hassani and Sh. Nasiri

Department of Computer Engineering, University of Sirjan Technology, Sirjan, Iran.

Received 17 January 2016; Revised 09 November 2016; Accepted 21 February 2017
**Corresponding author: pouramini@sirjantehc.ac.ir (A. Pouramini).*

Abstract
In this paper, we present an approach and a visual tool called Handle-based Wrapper (HWrap) for creating web wrappers to extract data records from web pages. In our approach, we rely mainly on the visible page content to identify the data regions on a web page. In our extraction algorithm, we were inspired by the way a human user scans the page content for a specific data. In particular, we use text features such as textual delimiters, keywords, constants or text patterns, which we call *handles*, to construct patterns for the target data regions and data records. We offer a polynomial algorithm, in which these patterns are checked against the page elements in a mixed bottom-up and top-down traverse of the DOM-tree. The extracted data is directly mapped onto a hierarchical XML structure, which forms the output of the wrapper. The wrappers that are generated by this method are robust and independent of the HTML structure. Therefore, they can be adapted to similar websites to gather and integrate information.

Keywords: *Web Data Record Extraction, Web Wrapper Generation, Web Information Extraction.*

1. Introduction

Extracting structured data from web pages has many applications in different areas including business and competitive intelligence, comparison shopping, customizable Web information gathering, and so on. [1]. Many research works have proposed methods to analyze web documents and extract their information in structured formats automatically; these proposals are commonly referred to as *information extractors* or *wrappers* [2-4]. These methods range from hard-coded wrappers to unsupervised wrapper induction methods. They vary mainly in the degree of automation they provide by reducing the human efforts. However, providing a higher automation can lead to a lower accuracy and a lesser flexibility [1].

Different approaches to wrapper generation model a web page in different ways. The most common approach is to work on the DOM-tree as the HTML structure of a document [2-4]. However, some other works claim that HTML is mainly used for the presentation layer. Therefore, it is not accurate enough to discriminate different semantic portions of a web document [8 ,10-12]. Moreover, as the complexity of typical web documents increases, information extractors have to analyze more and more irrelevant regions that have an impact on both efficiency and effectiveness [7, 10]. This has motivated a number of researchers to work on *region extractors* as a means to relieve information extractors from the burden of analyzing many regions of a web document that do not contain any relevant information [6-17].

From this viewpoint, a region is defined as an HTML fragment that shows information about an item or several related items when it is rendered on a web browser. Such items can be data records, e.g. information about products, goods, services or pieces of news, headers with navigation menus, footers with contact information or sidebars with advertisements, etc. The difference between region extractors and information extractors or wrappers is that the wrappers focus on extracting and structuring data records and their attributes, whereas region extractors focus on identifying the HTML fragments that contain this information.

In this paper, we present a supervised method to define patterns for the data regions on a webpage. Then we present an algorithm to apply the resulting patterns to a webpage.

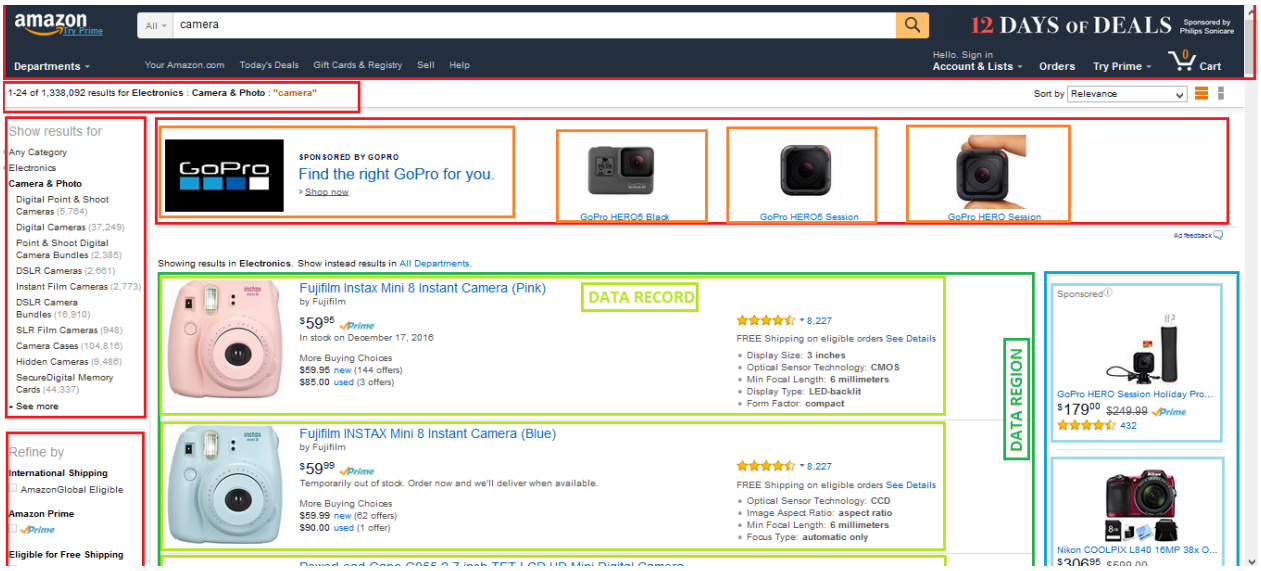

Figure 1. A sample results page of a website for "camera".

In our approach, we mainly rely on the page's visible content to locate the data regions and data records. As a result, the resulting wrappers are robust and easier to maintain. Moreover, they can be adapted to multiple websites with similar content structure to gather and integrate information from various sources. In the following sections, after reviewing the existing approaches, we present our data extraction method and an algorithm to implement it.

2. Related works

As mentioned before, a region extractor can be considered as a part of an information extractor or as a stand-alone application. Since in our method we focus on identifying data regions on a web page, we first review some proposals for extracting data regions from web documents.

Embley et al. [12] have proposed a method to extract the data records from the largest data region in a web document. It is an unsupervised method, which makes the following assumptions to identify data regions. There is a unique data region that is the largest region in the web document. This region contains multiple data records. Some tags are more likely to be data record separators based on their type and their occurrences. Finally, an ontology can help identify data records.

Some of the heuristics proposed by Embley et al. have been used in other region extractors [13]. For example, OMINI assumes that the main data region corresponds to the subtree with the largest number of children.

Mining Data Records [14] (or MDR for short) is another region extractor that aims to extract data records. It assumes that a data region contains repetitive structures in a document. Each repetitive structure inside a data region is a data record, and they are usually rendered inside tables and forms. There are other methods such as TPC and U-REST that search for repetitive and similar structures in a document to identify data regions [16, 17, 21]. Some of the assumptions they make include a data region containing multiple contiguous or non-contiguous data records. The data records have similar HTML structures, have small separators and are rendered similarly, visually aligned.

One of the most-cited region extractors is VIPS [8]. It is a vision-based approach to build the content structure of a web page by exploring the visual characteristic of the page elements and not only relying on their HTML hierarchical structure. The algorithm divides a web page into a collection of contiguous regions based on their visual properties. For example, if the background color of a child node is different from the background color of its parent, that child is counted as a sub-region. Figure 1 shows the regions returned by VIPS on a sample web page of Amazon website. Note that VIPS only identifies and separates regions in a web document; it is the user's responsibility to select the regions of interest (e.g. the green boxes in Figure 1).

Subsequently, some approaches to information extraction used the regions returned by VIPS as the basis for detecting and extracting informative regions. VSDR, ViDRE, and RIPB are some examples of such methods [9, 10, 15]. For example, RIPB [15] is a supervised method that requires a few examples of data records. Then a DOM-tree is built for each example using a tree alignment method.

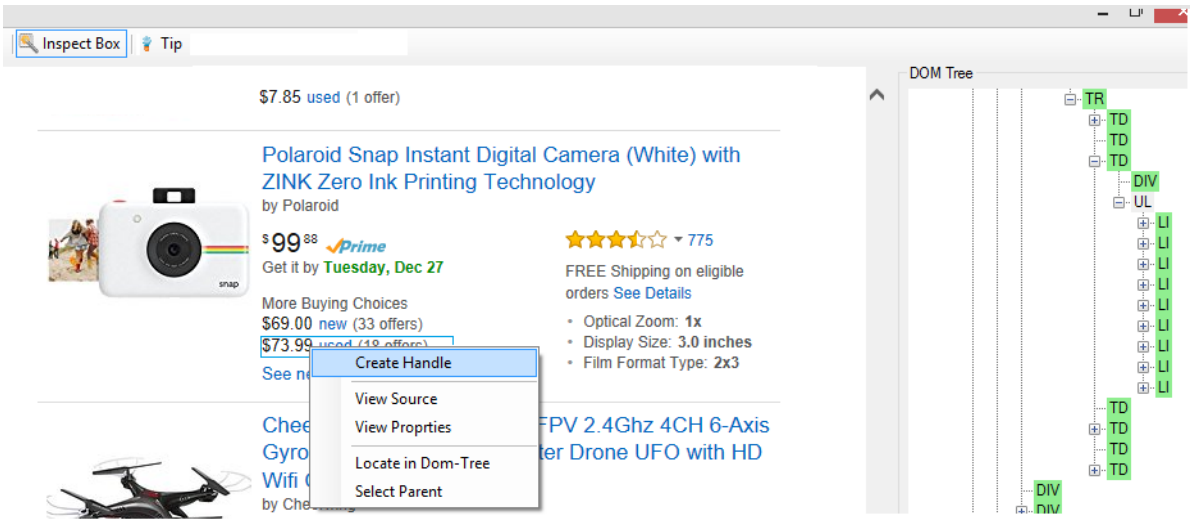

Figure 2. Identifying handles on a web page in our proposed system, HWrap.

These trees constitute tag patterns or the extraction rules. To extract data from an input document, the algorithm uses VIPS to segment the document into a collection of candidate regions. It then compares these regions with the tag patterns and returns the regions with the highest score. The similarity function is based on the tree-edit distance [18, 19].

In general, the proposed methods search for repetitive structures to identify data regions. As a result, they require the web page to contain at least two data records for the region extractor to work. Most of them are unsupervised and have assumptions about the structure of the data regions in a web document. Besides these assumptions about the layout of data regions, some may use an ontology [12, 13, 20]. While these unsupervised methods are scalable, they lack flexibility. Also, they may need many examples at the learning phrase. They often rely on the following algorithms to search a web document for data records: tree matching, string matching, and clustering [1, 18].

The majority of the proposed methods rely directly or indirectly on the DOM-tree or HTML tags. Some of them work on the region tree produced by VIPS [9, 10, 15]. This enables them to utilize visual information of the rendered page elements such as the position and the rendering box of each element to increase their accuracy. As a consequence, this makes it difficult to apply them to free-text documents whose contents do not rely heavily on HTML tags.

In the next section, we present our approach to identify data regions. We explain how our approach is different from the existing approaches.

3. Our approach

Figure 2 shows an example of a data-rich web page containing multiple instances of a data record. In such web pages, the HTML elements like table cells and divisions are frequently used to separate data records. Therefore, the DOM-tree is almost reliable to identify data regions. However, the main drawback of the approaches that mainly rely on the HTML structure is the lack of flexibility. They may use absolute HTML paths to locate an item. This approach is likely to fail when minor changes occur in the target HTML structure. Moreover, In the web of today, it is very common to use only DIV tags and describe the "semantics" of a particular division using style-sheet classes (CSS files). In this respect, the class name is perhaps the most notable semantic value among the element's attributes.

Our method basically relies on the DOM-Tree. However, we use the DOM-tree to make patterns based on delimiters and textual keywords in the content of a data record which is enclosed in one or more DOM-tree nodes. We assume that data regions are contiguous portions on a web page, comprised of one node or a range of nodes in the DOM-tree. To make the data region patterns independent of the DOM structure, we create them on the top of the page visible content. Our method is supervised, which means that we require the user to create the required patterns on a sample web page. However, as a future work, we aim to extract these patterns using some heuristics and an unsupervised learning.

In our method, we are inspired by the way people typically look for data on a web page. A human reader may scan a web page top-down or bottom-up, looking for signs to recognize the page structure. They rely on visual cues on the page (fonts, colors, text or link density) as well as semantic cues or text signals (titles, highlighted words, keywords, constants) to get a mental image

of the content structure. We refer to such textual signals with the term *handle* throughout this paper. A handle can be a visible element that marks the start or end of a data region or it can be a textual element or a regular expression in the visible text of a data region so that it distinguishes the data region from the rest of the page. In data-rich web pages, which are the main target of this work, such handles are prevalent. For example, figure 2 shows a web page containing the search results for a product, that includes multiple instances of the product. In such pages, there is usually the phrase "Search Results" or the pattern "number + 'results'" somewhere above a list of items. Such a phrase or text pattern based on it (regular expression) can establish a handle for identifying the start of a data region. As another example, in each product item, fields such as the product name, the price or the shipping details can establish handles to distinguish the product region. Such elements are not tailored to the template of the website, and thus they are less affected by modifications and revisions to the layout. In the next section, we explain how a wrapper can be built using handles.

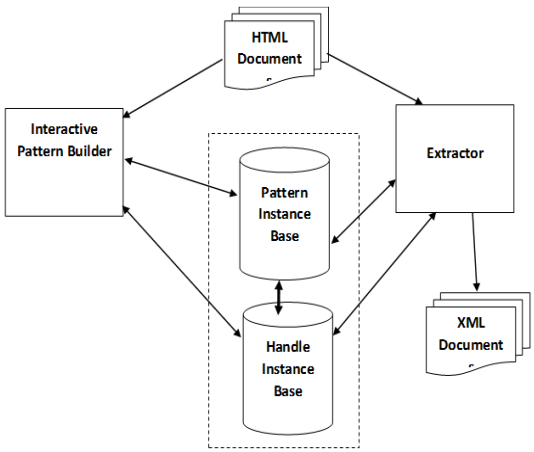

Figure 3. Architecture of HWrap.

3.1. Interactive wrapper generation

The overall architecture of HWrap wrapper generation toolkit is shown in figure 3. It consists of the following modules:

- The *Interactive Pattern Builder* provides the user interface that allows a user to visually specify the desired extraction patterns. The patterns are created by specifying one or more handles on a sample web page. The handles are saved into *Handle Instance Base*, and are indexed using a unique key. The created patterns are separately saved into another file called *Pattern Instance Base*, where they have some external references to the handles. These instance bases can be also stored as separate tables in a relational database.
- The *extractor* is the extraction engine that is provided with one or more web documents and the instance bases of handles and patterns. It identifies and extracts data records from the input documents and save them into separate XML documents, one per each input document. It can be used for extracting data from the pages of one or more websites that share common patterns within their content. In that case, the output of different documents can be integrated into a single output XML document.

In what follows, we describe the steps required for creating a wrapper in our proposed system.

3.1.1. Creating handles

The user loads a sample page into an embedded web browser to specify handles (see Figure 2). To facilitate this task, the page elements get highlighted as the user hover the mouse pointer over each element. After selecting an element, a pop-up allows the user for creating a handle based on some of the element's properties. This window is shown in figure 4 (Left). As seen in this figure, in addition to the text or a text pattern within the content, the user can create a handle using the values of some more constant attributes such as *id* and *class name* (e.g. the "*main*" value for the identifier of the main division of the page). These values can also be specified using a regular expression.

3.1.2. Creating region patterns

After creating handles, the user must specify how target data regions are identified by using these handles. The user creates region patterns based on one or more handles. Through this paper, we refer to such region patterns with the term *pattern*. Figure 4 (Right) shows different ways to create a pattern using the given handles.

For a single handle, the possible patterns are:

- *Self*: The region enclosed by the node that matches the handle.

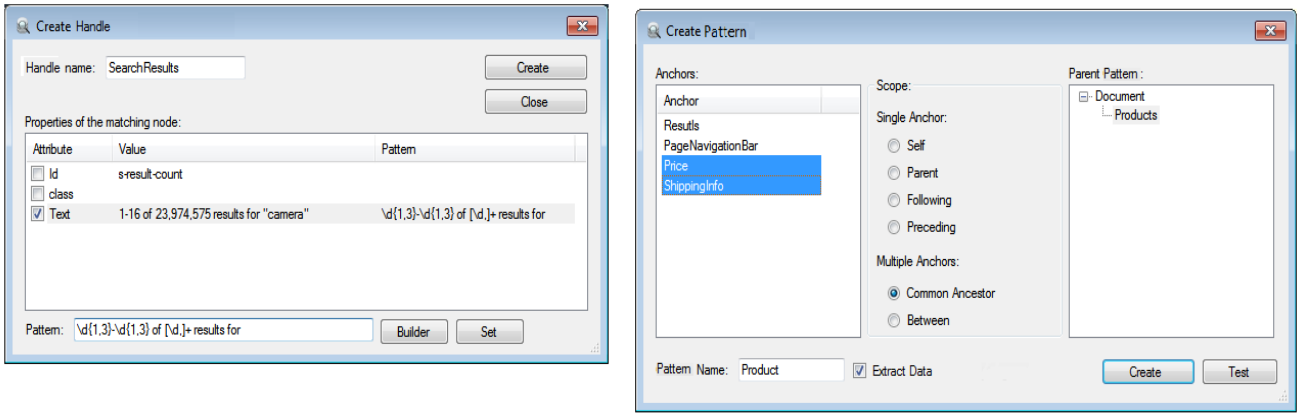

Figure 4. Screenshot of dialog boxes for creating handles (left) and patterns (right).

- *Parent*: The region enclosed by the handle's parent node (container node of the handle).
- Following: The region following the handle.
- Preceding: The region preceding the handle.

For multiple handles, a region pattern can be specified using the following options:

- *Common ancestor*: The region enclosed by the container of two or more handles.
- *Between*: The region that lies between two handles.

Each pattern can be designated either for data extraction purpose or for restricting the location of the other patterns. To illustrate how these patterns are used in the data extraction process, let us return to our previous example of a web page containing the results of a search for a product. Suppose we want to identify the region that encloses the list of the product items. We create a handle using an element containing the text "Search Results" to mark the beginning of this region. We then create the target pattern named "Products" using this handle and the "Following" option. This pattern matches a region containing the elements that follow the handle in the document. In order to restrict the length of this region, we can find another handle to mark the end of it. For example, "page navigation bar", which usually appears under a search results, is a suitable candidate for this purpose. Finally, the pattern can be created using these two handles and the "Between" option.
As another example, suppose we want to create a pattern for the region enclosing each product. We first identify two handles within a product item, such as the *price* and the *shipping details*. A handle for the price can be created using a text pattern that matches a price value and currency (e.g. $NN.NN). Similarly, for the shipping information, the handle can be created using the "Ship" or "Get it by" keywords, and a text pattern for a calendar date (e.g. "Get it by Month Day" or "Ships within N days"). Alternatively, any constant keyword repeating in all the product items, such as "Add to Cart", can be a candidate for creating a handle. Having these handles, we create the pattern "Product" as the common ancestor or the immediate container of these handles. To restrict the occurrence of this pattern to a certain part of the page, we can assign it a parent pattern from the list of the previously defined patterns (e.g. "Products").

3.2. Data extraction algorithm

In this section the data extraction algorithm is described. The handles and the patterns created in the previous section are input to this algorithm (see Algorithm 1). It is mainly a recursive function that is initially called with the root of the DOM-tree (body element) and traverses its nodes in depth-first manner. However, the algorithm may backtrack and travers a node for several times.
First, the input node is checked against the list of handles. If no match found, the function is recursively called with the child nodes. Otherwise, the pattern associated with the matching handle is retrieved from the list of patterns. Let's name this pattern *P*. If *P* has a parent pattern that has not been matched, it is ignored and the algorithm continues with the rest of the nodes. Otherwise, the state of *P* is updated to "Open" or "Closed" depending on the handle that has been matched. If *P* has been defined as a region over a node (Self, Parent and Common Ancestor options), the corresponding node in the DOM-tree is retrieved. Let's name this node *PNode*.

Algorithm 1 Data Extraction Algorithm

```
function Extract(node)
        handle ← MatchHandle(node, handles)
        if (handle !=null) then
            pattern ← GetPattern(handle, patterns)
            if (pattern = null or pattern.Parent ≠ "Open") then
                return null
            end if
            UpdatePatternState(pattern, handle)
            patternNode ← FindNode(pattern)
            if (pattern.ExtractDataFlag) then
                ExtractData (pattern, patternNode)
            end if
            if (patternNode ≠ null and pattern.Node ≠ patternNode) then
                ► the data region hasn't been previously evaluated
                pattern.Node ← patternNode
                if (pattern.hasChild) then
                    return pattern
                end if
            end if
    else
            for (i ← 0; i ≤ node.children; i ← i + 1) do
                child ← node.children[i]
                prevPattern ← curPattern
                curPattern ← Extract(child)
                if (prevPattern ≠ null and prevPattern.State = "Open") then
                    prevPattern.State ← "Closed"
                end if
                if (curPattern ≠ null and curPattern.Node ≠ null) then
                ► then the children of the pattern node musb be reevaluated
                    if (curPattern.Node = child) then
                        curPattern.State = "Open"
                        i ← i -1 ► decrement the counter to reevaluate the child
                    else
                        return curPattern ► return the pattern up the stack
                    end if
                end if
            end for
    end if
        return null
end function
```

PNode encloses a portion of the page which corresponds to the region specified by P. After matching *P*, if it has been designated for data extraction, its content is added to the output XML structure as an XML node with the same name as the *P*'s name. If *P* has a parent pattern, this node is placed under a node that represents its parent. Finally, if *P* has one or more child patterns, the function must reevaluate the *PNode*'s children once again to match the child patterns. To do this, the *P* instance including a pointer to *PNode* is returned to the function calling point in the "*for*" loop (line 24 in Algorithm 1). At this point, the algorithm steps back (in the case of Self) or traverse up (in the cases of Parent and Common Ancestor) to find the *PNode* in order to reevaluate its child nodes (see Algorithm 1 (continued)).

3.2.1. Time complexity

The time complexity of this algorithm is polynomial and depends on the size of the DOM-tree and the number of handles and patterns. In the best case, if the algorithm does not traverse back to revisit a node (traversing a node twice), then the time complexity is *O(N)*, where *N* is the number of the DOM-tree nodes. The explanation is as what follows. The algorithm is called *N* times, one call per each node of the DOM-tree. On each call, the input node is checked against the list of handles (*MatchHandle* at line 2), and if a match is found, a pattern is retrieved from the list of patterns (*GetPattern* at line 4). Searching through the list of handles and patterns depends on the number of handles, *H*, and the number of patterns, *P*. Therefore, *O(logH+logP)* is the total time for these operations. The runtime of other functions in the algorithm is roughly *O(1)*. The *FindNode* function at line 9 returns either the current node (in the case of the Self option) or an ancestor of the input node (in the cases of the Parent and Common ancestor options). It is supposed that the immediate ancestor of any two handle nodes, from which one is the current node, is in the first few levels directly above the current node in the DOM-tree. Also the *ExtractData* at line 16 receives a reference to the pattern node, and therefore, extracting its content takes *O(1)*. Since, in the best case, the number of patterns and handles is not

considerable, we can ignore the *(logH+logP)* term, and the time complexity is roughly *O(N)*.

In the worst case, when most of the patterns are defined using either "Parent" or "Common Ancestor" options, the algorithm's time complexity is $O(N^2 logN)$. The explanation is as what follows. In the worst case, for each node of the DOM-tree, at most one handle can be defined to match that node, and one pattern can be defined to match the region enclosed by this node. The algorithm, as before, is called *N* times. On each call, after matching a pattern, a sub-tree must be revisited, which in the worst case will be the entire DOM-tree; therefore, *N* is multiplied by *N* in the formula. *O(logH + logP)*, as before, is the time for searching through the lists of handles and patterns. By bounding *P* and *H* to *N*, the time complexity of these operations is *O(logN)*. The time complexity of the *FindNode* function is similarly *O(logN)* in the worst case because the algorithm must traverse the DOM-tree up to the root and it takes *O(logN)*. Therefore, the total time of the algorithm will be $O(N^2 logN)$, which is again polynomial. However, this is a very loose bound and in practice the time complexity is near to *O(N)*.

3.3. Data extraction and integration

Figure 5 displays the result of applying the extraction algorithm to the web page shown in fig. 2. In this structure, each pattern forms an XML node, which is nested in another node that corresponds to its parent. If the user desires to extract any specific data, he must first create an appropriate pattern for the region that encloses that data and mark it for data extraction ("Extract Data" checkbox in the form shown in Figure 4). As seen in fgure 5, the exact HTML source code of each product item is added to the "Product" node in the output XML tree. This code can be further processed to extract the subfields (e.g. price, bids, and supplier) or be directly rendered in a web browser to be displayed to the user for visual comparison.

If the user defines a pattern more generally by ignoring details, the resulting wrapper can be reused for gathering and integrating data from different websites that have this pattern in common. In some cases, minor changes are required to adapt an existing handle or pattern to a new website. Therefore, the user often needs to follow the same procedure to create wrappers for similar websites, and this makes creating them easier. To facilitate this process, the Pattern Builder automatically highlights the matching items on the page when the user loads a previously created pattern file.

On the other hand, the user can define a pattern in more details when he wants to extract some fields of a data record. To do so, he must create a pattern for each target filed inside the pattern that has been defined for the data record. For example, to extract the price field from each product and store it as a child node of the product's node in the XML tree, an appropriate pattern can be defined using the "price" handle, the "Self" option, and the "Products" pattern as its parent.

4. Experimental results

We tested the accuracy and expressiveness of our proposed method on a number of websites including Amazon, eBay, IMDB, world weather, and YouTube (see Table 1). We collected 30 pages from each website. Table 1 shows the number of handles and patterns required to identify the data records on each website.

In all cases, the website was wrapable and required a few number of handles to identify the data records. Similar to the example reviewed through this paper, the handles were easily identifiable on a sample web page. The "Common ancestor" option was very useful to specify a record by identifying two or more handles within it. The experiments show that by providing suitable handles and patterns, the system can achieve a high performance of 98.9% in F-Measure.

Table 1. Evaluation of generated wrappers.

Website	# Handles	# Patterns	Precision	Recall	F-Score
Amazon	5	3	98.5	100	99.2
eBay	5	3	98.7	100	99.3
IMDB Search	5	4	97.6	100	98.7
World Weather	4	2	98.4	99.5	98.9
Youtube	4	3	97.3	98.9	98.1
Average			**98.1**	**99.7**	**98.9**

The precision refers to the average fraction of regions that are identified by the system and correspond to the actual data regions. Recall refers to the average fraction of actual data regions that are identified by the system. The higher recall values in table 1 indicate that most of the actual data regions were identified by our method. However, the lower performance values indicate that some of the identified regions are not the actual data regions. For example, they can be some records on the advertisement area beside the main data region (see Figure 1).

The performance of our proposed system is close to the performances reported by the related works in this filed. Table 2 shows the performance

measures reported by some of the proposals that were reviewed in the "Related Works" section.

Table 2. Performance of some related works.

Proposal	Precision	Recall
OMINI	100	94
MDR	100	99.8
RIPB	98.1	95.7
TPC	96.	97.0
VSDR	89	97.6

Note that they are not comparable side-by-side because they were calculated on different datasets. However, they can provide a rough reference for comparison. As is seen, the performance of our proposed method, especially in terms of recall, is higher than most of these proposals. One reason is that most of these methods are unsupervised and search for repetitive structures within the page. Therefore, when a data record has a slightly different structure than the rest, it can be ignored. Since our method relies on the content of the data records rather than their internal structures, it can better cover such structural dissimilarities in data records.

Table 3 shows the percentage of the handles that are common among the websites used in the experiment. Since our method relies on the visible content and textual identifiers, common handles can be found in the websites that offer a similar service. For example, in most shopping websites, the *price* and *shipping details* fields are among a product's information, and thus they can be used to identify a product item by selecting "Common Ancestors" Option. Note that this option finds the closest ancestor that encloses all the specified handles regardless of the specific structure of each website and the number of nesting elements.

Table 3. Shopping websites with similar handles.

Website	Common Handles
eBay	80%
DHGate	80%
AliExpress	80%
Etsy	50%

5. Conclusion and future works

In this paper, we presented an approach to the problem of extracting data records from web pages. We based our approach on the textual handles within the visible content of the web page. We were inspired by the way a human user scans a web page for the data of interest. Handles serve as identifiers for a data region so that a pattern can be constructed on the basis of one or more handles. Each pattern may be treated as the sub-pattern of another pattern. Given these patterns, we proposed a data extraction algorithm. This algorithm traverses the DOM-tree nodes in a mixed top-down and bottom-up manner to match the given handles and patterns. This algorithm is time-polynomial and in the worst case, has $O(N^2 logN)$ time complexity. However, in the average case, it grows linearly with the number of nodes $(O(N))$. As our proposed method relies mainly on the page visible content rather than the HTML structure, the generated wrappers are robust and maintainable. We showed that they could be adapted to gather data from similar web pages and integrate them into an XML document.

In this paper, we defined a pattern for a data region using one or several handles. In addition, we provided a means to restrict the location of a pattern inside another pattern. As a future work, we aim to provide an option to define a pattern based on handles and other patterns. For instance, a pattern can be defined as the common ancestor of two existing patterns or one handle and one existing pattern. For example, the data region enclosing several product records can be defined as the common ancestor of two product patterns.

As another work, we aim to find the required handles and patterns on a web page using an unsupervised method similar to the works reviewed in the "related works" section. Then we use these patterns in our proposed algorithm to extract the data records.

References

[1] Ferrara, E., De Meo, P., Fiumara, G., & Baumgartner, R. (2014, November). Web data extraction, applications and techniques: A survey. Knowledge-Based Systems, vol. 70, no. 1, pp. 301-323.

[2] Sahuguet, A., & Azavant, F. (1999, September). Building light-weight wrappers for legacy web data-sources using W4F. In Proceeding of VLDB, pp. 738-741.

[3] Liu, L., Pu, C., & Han, W. (2000). XWRAP: An XML-enabled wrapper construction system for web information sources. In Data Engineering Proceedings. 16th International Conference on (pp. 611-621). IEEE.

[4] Gottlob, G., Koch, C., Baumgartner, R., Herzog, M., & Flesca, S. (2004, June). The Lixto data extraction project: back and forth between theory and practice. In Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (pp. 1-12). ACM.

[5] Wang, J., & Lochovsky, F. H. (2003, May). Data extraction and label assignment for web databases. In

Proceedings of the 12th international conference on World Wide Web (pp. 187-196). ACM.

[6] Bing, L., Lam, W., & Gu, Y. (2011, October). Towards a unified solution: data record region detection and segmentation. In Proceedings of the 20th ACM international conference on Information and knowledge management (pp. 1265-1274). ACM.

[7] Wang, J., & Lochovsky, F. H. (2002, December). Data-rich section extraction from html pages. In Web Information Systems Engineering, 2002. WISE 2002. Proceedings of the Third International Conference on (pp. 313-322). IEEE.

[8] Cai, D., Yu, S., Wen, J. R., & Ma, W. Y. (2003). VIPS: A vision-based page segmentation algorithm. Microsoft technical report, MSR-TR-2003-79

[9] Liu, W., Meng, X., & Meng, W. (2006, July). Vision-based web data records extraction. In Proc. 9th international workshop on the web and databases (pp. 20-25).

[10] Li, L., Liu, Y., Obregon, A., & Weatherston, M. (2007, August). Visual segmentation-based data record extraction from web documents. In Information Reuse and Integration, 2007. IRI 2007. IEEE International Conference on (pp. 502-507). IEEE.

[11] Li, L., Liu, Y., Obregon, A., & Weatherston, M. (2007, August). Visual segmentation-based data record extraction from web documents. In Information Reuse and Integration, 2007. IRI 2007. IEEE International Conference on (pp. 502-507). IEEE.

[12] Embley, D. W., Jiang, Y., & Ng, Y. K. (1999, June). Record-boundary discovery in Web documents. In ACM SIGMOD Record (vol. 28, no. 2, pp. 467-478). ACM.

[13] Buttler, D., Liu, L., & Pu, C. (2001, April). A fully automated object extraction system for the World Wide Web. In Distributed Computing Systems, 2001. 21st International Conference on. (pp. 361-370). IEEE.

[14] Liu, B., Grossman, R., & Zhai, Y. (2004). Mining web pages for data records. IEEE Intelligent Systems, vol. 19, no. 6, pp. 49-55.

[15] Kang, J., & Choi, J. (2008). Recognising Informative Web Page Blocks Using Visual Segmentation for Efficient Information Extraction. J. UCS, vol. 14, no. 11, pp. 1893-1910.

[16] Shen, Y. K., & Karger, D. R. (2007, May). U-REST: an unsupervised record extraction system. In Proceedings of the 16th international conference on World Wide Web (pp. 1347-1348). ACM.

[17] Miao, G., Tatemura, J., Hsiung, W. P., Sawires, A., & Moser, L. E. (2009, April). Extracting data records from the web using tag path clustering. In Proceedings of the 18th international conference on World wide web (pp. 981-990). ACM.

[18] Zhai, Y., & Liu, B. (2005, May). Web data extraction based on partial tree alignment. In Proceedings of the 14th international conference on World Wide Web (pp. 76-85). ACM.

[19] Bille, P. (2005). A survey on tree edit distance and related problems. Theoretical computer science, vol. 337, no. 1, pp. 217-239.

[20] Su, W., Wang, J., & Lochovsky, F. H. (2009). ODE: Ontology-assisted data extraction. ACM Transactions on Database Systems (TODS), vol. 34, no. 2, pp. 12-17.

[21] Naeem, M., Bilal Khan, M., & Tanvir Afzal, M. (2013). Expert Discovery: A web mining approach Journal of AI and Data Mining: Shahrood University of Technology, vol. 1, no. 1, pp. 35-47.

نشریه هوش مصنوعی و داده کاوی

استخراج داده از وب با استفاده از دستگیره‌های مبتنی بر محتوی

احمد پورامینی*، سمیه خواجه حسنی و شهرام نصیری

گروه برق و رایانه، دانشگاه صنعتی سیرجان، سیرجان، ایران.

ارسال ۲۰۱۶/۰۱/۱۷؛ بازنگری ۲۰۱۶/۱۱/۰۹؛ پذیرش ۲۰۱۷/۰۲/۲۱

چکیده:

در این مقاله، ما روشی را با نام پوشنده مبتنی بر دستگیره[۱] (HWrap) برای ایجاد پوشنده‌های وب جهت استخراج فقره‌های داده از صفحات وب ارائه می‌کنیم. در این رویکرد، برای تشخیص نواحی حاوی داده از محتوای قابل مشاهده صفحه استفاده می‌کنیم. الگوریتم استخراج داده از نحوه پویش یک صفحه توسط انسان برای جستجوی اطلاعات موردنظر الهام گرفته است. به طور مشخص، ما از برخی ویژگی‌های متنی مانند جداسازهای متنی، کلمات کلیدی، ثابت‌ها یا الگوهای عددی و متنی که به آنها دستگیره می‌گوییم جهت ایجاد الگوهایی برای نواحی حاوی داده و فقره‌های داده‌ای استفاده می‌کنیم. سپس یک الگوریتم با پیچیدگی چندجمله‌ای برای استخراج داده با استفاده از این الگوها ارائه می‌کنیم. در این الگوریتم، در یک پیمایش توأم بالابه‌پایین و پایین‌به‌بالا الگوهای معرفی شده با ساختار Html (درخت DOM) مطابقت داده می‌شوند و ساختارهای منطبق با این الگوها استخراج می‌شوند. سپس داده‌های استخراج شده مستقیماً به یک ساختار سلسله‌مراتبی XML متناظر با الگوهای تعریف شده نگاشته می‌شوند. پوشنده‌ای که با این روش ساخته می‌شوند مبتنی بر محتوای صفحات بوده و بسیار باثبات هستند. همینطور می‌توان از آنها در وب‌سایتهای که محتوای مشابهی برای فقره‌های داده‌ای دارند استفاده کرد.

کلمات کلیدی: داده کاوی، وب کاوی، استخراج اطلاعات، استخراج فقره داده از وب، پوشنده وب.

[۱] Handled-based Wrapper