GenDE: A CRF-Based Data Extractor

Mohammed Kayed[1,*] and Khaled Shaalan[2]

[1]*Faculty of Computers and Artificial Intelligence, Beni-Suef University, Beni-suef, Egypt, 62511*
[2]*Faculty of Engineering and IT, The British University in Dubai, Dubai, UAE*
E-mail: mskayed@gmail.com; Khaled.shaalan@buid.ac.ae
[*]*Corresponding Author*

Received 01 October 2019; Accepted 23 April 2020;
Publication 12 June 2020

Abstract

Web site schema detection and data extraction from the Deep Web have been studied a lot. Although, few researches have focused on the more challenging jobs: wrapper verification or extractor generation. A wrapper verifier would check whether a new page from a site complies with the detected schema, and so the extractor will use the wrapper to get instances of the schema types. If the wrapper failed to work with the new page, a new wrapper/schema would be re-generated by calling an unsupervised wrapper induction system. In this paper, a new data extractor called GenDE is proposed. It verifies the site schema and extracts data from the Web pages using Conditional Random Fields (CRFs). The problem is solved by breaking down an observation sequence (a Web page) into simpler subsequences that will be labeled using CRF. Moreover, the system solves the problem of automatic data extraction from modern JavaScript sites in which data/schema are attached (on the client side) in a JSON format. The experiments show an encouraging result as it outperforms the CSP-based extractor algorithm (95% and 96% of recall and precision, respectively). Moreover, it gives a high performance result when tested on the SWDE benchmark dataset (84.91%).

Keywords: Wrapper induction, data extractor, wrapper verifier, sequence labeling, CRFs model, JSON data extraction.

Journal of Web Engineering, Vol. 19_3-4, 371–404.
doi: 10.13052/jwe1540-9589.19342

1 Introduction

The Deep Web, which contains between 400 and 550 times more public information than the Surface Web, is generated dynamically using some predefined templates embedded by data taken from databases. Although many of these embedded data are publicly available as Web services, many others are still not accessible through a programming interface. So, developing a wrapper to extract such embedded data is useful for many Web intelligent systems. Wrapper induction (WI) which aims to generate extraction rules (wrappers) for Web data extraction is a key component for many sophisticated Web mining applications. Many WI approaches are surveyed in [1]. These approaches have different extraction targets/levels (field-level [2], record-level [3] [4] or page-level [5] [6]), various automation degree (supervised or unsupervised) and different features for rule representation (HTML tokens, DOM tree, visual features, etc.). As the schema and template of a dynamic Web site are usually changed with the time, researchers in the domain of page-level Web data extraction aim not only to construct the Web site schema and extract data based on this schema, but also to generate a "wrapper maintenance" tool that will validate the constructed schema before extracting data at a later time. The wrapper maintenance tool will check whether either the site schema or the site template are changed or not. If no changes has been occurred, the data extractor starts its extraction job, otherwise a wrapper re-induction step must be done to get the new site schema or template.

This paper addresses two main shortages in the current Web data extraction approaches. First, all current approaches consider data to be extracted from a Web site as embedded in the HTML tags of the site pages, and then mine these pages (DOM trees) to detect the Web site schema and extract the data (i.e., data and schema are unknown). Although, these approaches ignore an important fact that both data and schema may be partially or completely attached (on the client side) in modern JavaScript Web applications in a structured format such as JSON. Once data to be extracted are identified and accessed on the client side in a JSON format, the schema will be directly generated as JSON data are represented as name/value pairs. Furthermore, data/schema are labeled. Generally in a Web site, data to be extracted and its schema are provided in either of the following three different manners:

- Data and schema are only embedded in the pages' DOM trees (no JSON data),
- Data to be extracted are completely provided in a JSON format and the DOM trees as well (full JSON data),

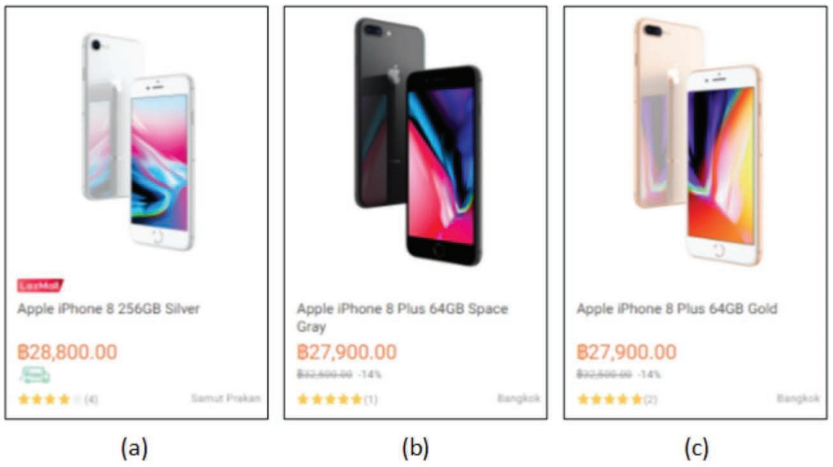
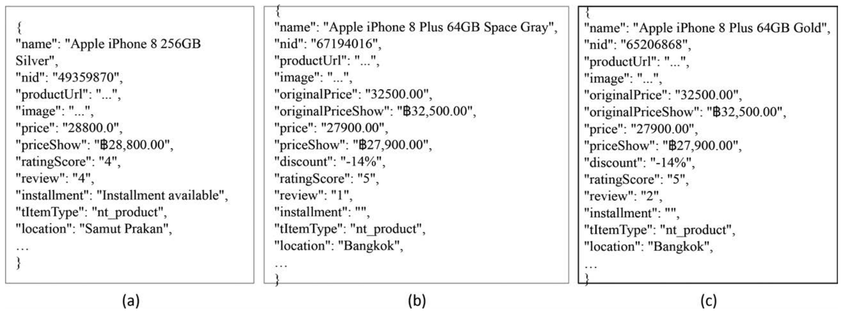


Figure 1 A part of LAZADA Web page (3 data records (a), (b) and (c)).



```
{
"name": "Apple iPhone 8 256GB
Silver",
"nid": "49359870",
"productUrl": "...",
"image": "...",
"price": "28800.0",
"priceShow": "฿28,800.00",
"ratingScore": "4",
"review": "4",
"installment": "Installment available",
"tItemType": "nt_product",
"location": "Samut Prakan",
...
}
```

```
{
"name": "Apple iPhone 8 Plus 64GB Space Gray",
"nid": "67194016",
"productUrl": "...",
"image": "...",
"originalPrice": "32500.00",
"originalPriceShow": "฿32,500.00",
"price": "27900.00",
"priceShow": "฿27,900.00",
"discount": "-14%",
"ratingScore": "5",
"review": "1",
"installment": "",
"tItemType": "nt_product",
"location": "Bangkok",
...
}
```

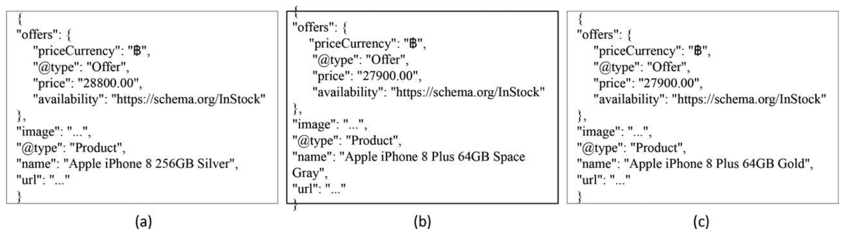
```
{
"name": "Apple iPhone 8 Plus 64GB Gold",
"nid": "65206868",
"productUrl": "...",
"image": "...",
"originalPrice": "32500.00",
"originalPriceShow": "฿32,500.00",
"price": "27900.00",
"priceShow": "฿27,900.00",
"discount": "-14%",
"ratingScore": "5",
"review": "2",
"installment": "",
"tItemType": "nt_product",
"location": "Bangkok",
...
}
```

Figure 2 A full JSON objects ((a), (b) and (c)) corresponding to the 3 records in Figure 1((a), (b) and (c), respectively).

- Data to be extracted are partially provided in a JSON format, while all data are embedded in the DOM trees (partial JSON data).

Examples of these three types of Web sites are AMAZON[1], LAZADA[2] and SOUQ[3], respectively. Figure 1 shows a part (3 data records: (a), (b) and (c)) of a Web page taken from Lazada online shopping site. The whole data of this Web page are attached in a JSON format on the client side. Figure 2((a), (b) and (c)) shows the JSON data (objects) corresponding to the three data

[1]https://www.amazon.com/
[2]https://www.lazada.com/
[3]https://egypt.souq.com/eg-en/



```
{
"offers": {
    "priceCurrency": "฿",
    "@type": "Offer",
    "price": "28800.00",
    "availability": "https://schema.org/InStock"
},
"image": "...",
"@type": "Product",
"name": "Apple iPhone 8 256GB Silver",
"url": "..."
}
```

(a)

```
{
"offers": {
    "priceCurrency": "฿",
    "@type": "Offer",
    "price": "27900.00",
    "availability": "https://schema.org/InStock"
},
"image": "...",
"@type": "Product",
"name": "Apple iPhone 8 Plus 64GB Space
Gray",
"url": "..."
}
```

(b)

```
{
"offers": {
    "priceCurrency": "฿",
    "@type": "Offer",
    "price": "27900.00",
    "availability": "https://schema.org/InStock"
},
"image": "...",
"@type": "Product",
"name": "Apple iPhone 8 Plus 64GB Gold",
"url": "..."
}
```

(c)

Figure 3 A partial JSON objects ((a), (b) and (c)) corresponding to the 3 records in Figure 1((a), (b) and (c), respectively).

records shown in Figure 1((a), (b) and (c), respectively). The three objects in the figure and other similar objects (about 40 objects) are provided as a list ([]) with label "listItems". For simplicity, we remove part of these JSON data objects and replace it by "..." to save space. All data displayed in Figure 1 and more are shown in the JSON format in Figure 2. Furthermore, these data are labeled (e.g., "name", "price", etc.). Sometimes, data to be extracted are partially provided in a JSON format in the Web page (for instance SOUQ online shopping site). LAZADA Web site is chosen because it provides the two types of JSON data at the same time. Figure 3((a), (b) and (c)) gives the partial JSON data objects corresponding to the three data records shown in Figure 1((a), (b) and (c), respectively). To the best of our knowledge, no prior work has been proposed to automatically extracting JSON data from a Web site ([7] suggested an approach that did that based on some extraction rules and configurations defined manually).

Second, although many researches focused on wrapper induction, the development of tools for wrapper maintenance (wrapper verification and wrapper re-induction [8]) has received less attention. The wrapper maintenance problem becomes important because most web sites are always dynamic, not static. Hence, tracking the correctness of a wrapper and repairing it (when the schema is modified) have been a critical problem in practical applications. Researchers thought that unsupervised Web data extraction approaches don't need to maintain their wrappers as these approaches are designed to extract data without annotated training pages and can be used to extract data for each test page. Thus, wrapper verification is not needed for the unsupervised approaches, but schema matching is required to ensure data extracted at different time are well corresponded as shown in Figure 4. However, the process of generating a wrapper (extractor) and detecting a schema in the unsupervised approaches usually takes longer time (several seconds) than the process of using the generated wrapper and schema to

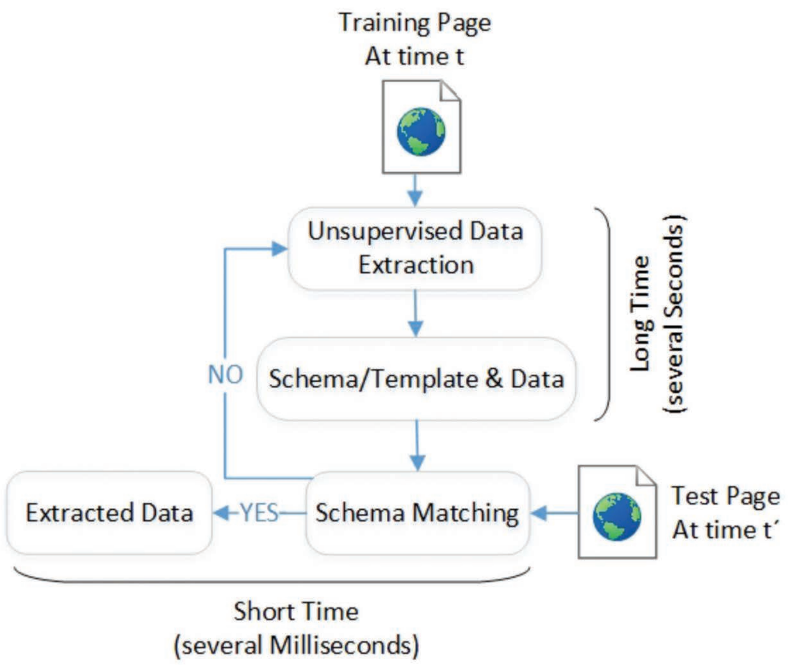


Figure 4 Schema matching for unsupervised Web data extraction approaches.

extract data (several milliseconds). Also, schema matching is a challenge for two schemas without labels. Therefore, both wrapper verification and schema matching for unsupervised Web data extraction systems are still of great necessity.

To the best of our knowledge, different from our proposed approach which is working with a page-level WI system, most prior wrapper maintenance approaches have been worked with record-level WI systems (e.g., [9], [10] and [11]). Two prior wrapper maintenance approaches are designed to work with page-level WI systems: Constrained Satisfaction Problem (CSP) based approach [12] and the Joint Wrapper and Data Repair (WADaR) approach [13] [14]. The former one constructs a finite state machine (FSM) by using the schema detected and the data extracted at a specific time t. This constructed FSM is then used to verify the schema and extract data at a later time t'. But, the problem in this approach is that real Web sites usually have complex schemas. So, generating a FSM from such complex schemas is impractical and not effective. The later approach, WADaR, aligns data extracted by a WI system to a target schema using a “Fitness” function which is calculated based on missed, misplaced or under segmented data instances. This approach has an assumption that, given a target schema at time t, data extracted at a later time t' using the wrapper are correct but need to realigned which is incorrect in many real cases. As many Web sites are dynamic, so the wrapper may totally extract wrong data or even fail to extract any data when the site schema is changed.

Therefore, this paper proposes a Generic Data Extractor (GenDE) that tries to handle the two shortages mentioned above and so it is able to verify/extract data from JavaScript Web sites (contain JSON data) as well as the other traditional Web sites (do not contain JSON data). Our proposed wrapper maintenance approach uses the CRFs model for schema verification and data extraction. To the best of our knowledge, CRFs hasn't been used for wrapper maintenance (it is only be used for wrapper induction). We don't choose to solve this problem using the recent deep learning sequence model Recurrent Neural Network (RNN) which outperforms the traditional CRFs model in many domains. As RNN requires big training data to learn, although data available for wrapper maintenance is small (the data extracted previously by an unsupervised WI system using just few pages).

The rest of the paper is organized as follows. Our research problem is formulated in Section 2. Related works are presented in Section 3. The details of the proposed system are given in Section 4, while the experimental results are shown in Section 5. Finally, Section 6 concludes our work.

2 Problem Formulation

Web data extractor is considered as a reverse engineering tool for the Deep Web in which data are taken from a database, embedded into a predefined template T and submitted to users (in the HTML format) in response to their requests. Data instances in the database conform to a common schema which could be represented in an XML, JSON or any other format. Data to be extracted are allocated in the leaf nodes (text/image) while composite data (e.g. tuple, set, option, etc.) are allocated in the internal nodes of the site pages. Figure 5(a) shows the schema of the data records (instances) shown in Figure 1, where it has one set/list type ($[]_1$), three tuples ($<>_2$, $<>_4$, $<>_8$) and seven basic types (β_3, β_5, β_6, β_9, β_{10}, β_{11}, β_{12}) such that the two types (β_9, β_{10}) are optional ($()?_7$). This kind of Web site schema could be generated automatically by unsupervised WI systems such as FiVaTech [5] [15] and TEX [6]. We call this schema "DOM Schema". Another simpler but rich schema (called "JSON schema") could be generated directly by a simple

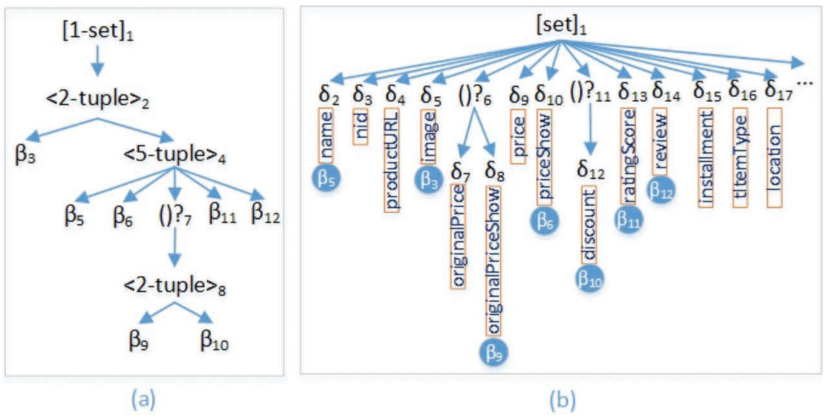


Figure 5 A DOM schema (a) and JSON schema (b) for the data shown in Figures 1 and 2.

algorithm from JSON data. In this JSON schema, sets (lists) are denoted by brackets ([]), tuples are denoted by braces () and basic types are denoted by δ_i $(i = 2, 3, ...)$. An instance of the basic type is a name/value pair, where value may be string, number, true/false or null (optional). This means, JSON schema basic types are labeled. Figure 5(b) shows the schema of the JSON data in LAZADA Web site shown in Figure 2. Note that, δ_2 has a label "name" and three instances "Apple iPhone 8 256GB Silver", "Apple iPhone 8 Plus 64GB Space Gray" and "Apple iPhone 8 Plus 64GB Gold" in the three JSON data objects (records) in Figure 2. The label is shown in Figure 5(b) inside a text box beneath each JSON basic type.

Definition 1 (Co-Basic Type). A basic type δ in the JSON schema is called a co-type if there exists a basic type β in the DOM schema such that the two types δ and β have the same instance values. Each of the two types δ and β would be called a co-type of the other.

As an example, the type δ_2 (with label "name") has a corresponding basic type β_5 in the DOM schema, so δ_2 and β_5 are co-basic types. Also, the types $\delta_5, \delta_8, \delta_{10}, \delta_{12}, \delta_{13}, \delta_{14}$ are co-basic types of the basic types $\beta_3, \beta_9, \beta_6, \beta_{10}, \beta_{11}, \beta_{12}$, respectively (a co-basic DOM schema type is shown inside a filled circle in Figure 5(b) beneath each JSON basic type). The other JSON schema basic types δ_3 (with label "nid"), δ_4 (with label "productURL"), δ_7 (with label "originalPrice"), δ_9 (with label "price"), δ_{15} (with label "installment"), δ_{16} (with label "tItemType") and δ_{17} (with label "location") have no corresponding basic types in the DOM schema, so they are not co-basic types.

Definition 2 (Partial/Full JSON Schema). If each basic type (β) in the DOM schema is a co-basic type for some basic type δ in the JSON schema, we call this JSON schema full, otherwise we call it a partial JSON schema. This means, a Web site has a full JSON schema if all data displayed by the Web browser attached in some JSON objects. For example, the schema shown in Figure 5(b) is a full JSON schema.

Our proposed generic data extractor GenDE is divided into two main phases: schema detection (both DOM and JSON schemas) and wrapper verification/extraction. In the first phase, given some input pages from a Web site, GenDE uses the unsupervised page-level Web data extraction system FiVaTech to generate the DOM schema of the site. Also, a JSON parser along with a simple algorithm are used to directly generate the JSON schema (if there is a full or partial one) as we shall discuss later. Finally in this phase, the generated DOM schema is updated (co-basic types are marked in the DOM schema) using the JSON schema to get a generic schema. In the second phase, a wrapper verifier will decide whether an input test page follows the template and the constructed schema; and if the answer is “Yes”, an extractor will extract data instances for each basic type.

3 Related Works

The Conditional Random Field (CRF) model is widely used to solve many problems in different applications such as speech, audio and language processing [16]. Many researchers have used CRF in the field of addresses extraction [17] and Web information extraction in general. Different from our proposed approach that uses CRF for page-level data extraction and verification, almost all other approaches adapted CRF to solve the record-level data extraction [15, 16] ignoring the maintenance problem. [18] used an existing record-level WI system to segment the Web pages into object blocks that are further segmented into atomic extraction entities called object elements. It suggested a two dimensional (2D) CRF model to label these elements. The 2D CRF model assumed that the object (record) elements have a two dimensional grid according to their position and size information, then associated each element with a state (hidden label). [19] used some heuristics to divide the pages into object blocks (data records), then used Information Entropy to filter out noisy blocks. It then used the edit distance to identify similar blocks and finally applied CRF to label (extract elements) from these blocks.

Wrapper maintenance (wrapper verification and wrapper re-induction) has received less attention from most researchers in the field of Web data extraction which has been widely studied from various aspects [1]. Most of the studies have focused on wrapper maintenance for record-level wrappers. RAPTURE [8] [9] [20] is an example of such record-level wrapper verifiers which exploited nine features: digit density, letter density, upper-case density, lower-case density, punctuation density, HTML density, length, word count, and mean word length to measure similarities between previously observed data by the wrapper and the data to be extracted in the future. As another example of a record-level wrapper verification is DATAPROG [10], in which two types of features from the previously extracted data are used: the learned patterns by DATAPROG (such as common beginnings or ending of the specific data fields) and the global numeric features. For each data field in the record to be extracted, the algorithm of wrapper verification decided whether an instance of this field is correct by statistically comparing training instances and the new data instance. If they are statistically similar, the wrapper is considered as correct. In fact, the verification of data semantic is not necessary if the extraction process is not complete or failed. Therefore, Pek et al. [11] introduced an adjacency-weight method to validate Web wrappers. The method used three features in the validation of the wrapper: HTML tree paths, number of children nodes per parent node and the number of possible data nodes that are extractable using this extraction path. If the number of extracted leaf nodes is less than or equal to the number of previously extracted nodes, the wrapper is considered as valid. Otherwise, it is considered as invalid. In a way, the consideration based on DOM tree is complementary to RAPTURE which is based on data semantic. This means both RAPTURE and DATAPROG tracked the semantic changes of data instances using the similarity of two distributions, while Pek et al. tracked the structural changes of the page DOM tree. The semantic features are used by RAPTURE and DATAPROG because data may be extracted successfully but it is semantically incorrect. If the extractor is failed, there is no need for semantic verification. Instead, wrapper re-induction is required.

The problem of computing joint repairs for wrappers and their extracted data (wrapper maintenance) has been studied in WADaR [13]. Given as input data generated by a wrapper in a structured format (a set of rows where each row is a sequence of instance values for specific attributes), this approach modified these data as follows. Misplaced data are handled by re-arranging it using an oracle function, which could also be replaced by an ensemble of entity recognizers that tagging data instances with types corresponding to the schema attributes. Also, data instances are re-segmented or merged to handle under or over segmentation, respectively. Further, unwanted data are removed. Finally, the main aim of this study is to align the relation to a target schema, and then adjusting the wrapper as much as possible. It quantified the agreement between a relation and a schema using a "Fitness" function which is based on the amount of misplaced or under segmented data instances. A CSP-based wrapper verification for page-level Web data extraction is proposed in [12]. The verification is done in consecutive phases based on the previously extracted data and the constructed schema/template. These phases are path-guided verification, data semantic verification and schema verification. Schema verification is done using a finite state machine, which is further used for data extraction. If a verification is failed at any of these phases, the unsupervised wrapper induction would be called for wrapper re-induction.

4 GenDE: The Proposed Approach

To avoid the threads to validity of Web data extraction approaches due to the dynamic nature of Web sites (schemas and templates are changed with time), wrapper verification and data extraction are combined to produce a comprehensive data extraction system (GenDE). Moreover, GenDE assumes that a Web site schema and data to be extracted are either attached in a JSON format provided by the Web pages on the client side or embedded in the pages DOM tress. As mentioned before and shown in Figure 6, the proposed system GenDE has two phases. The aim of the first phase is constructing the (JSON/DOM tree) schema and extracting data from the pages input to the unsupervised WI system FiVaTech at time t. Later at a different time t' in the second phase, given an input page from the site, GenDE includes two main alternative steps. First, if the generated schema is a full JSON schema, it will use a JSON verifier to verify and extract data from the embedded JSON data (see left hand side of Figure 6). Otherwise, the second step will apply the two modules "Template Verifier" and "CRF-Based Verifier" for the template verification and the semantic/schema verification of the input page, respectively. In this section, we shall discuss the two phases in details.

4.1 Generic schema construction

As shown in Figure 7, a core step to generate the schema of a Web site given some pages as input is constructing the DOM schema using an unsupervised

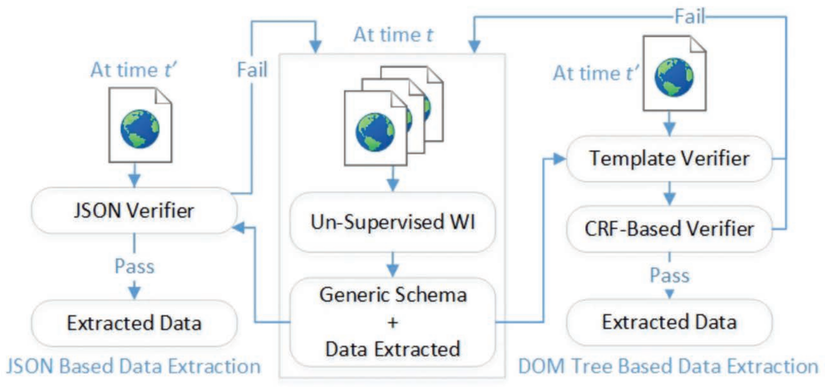


Figure 6 The structure of the proposed GenDE data extractor.

page-level Web data extraction system. GenDE uses FiVaTech which constructs the DOM schema of the Web site and extract data from the input pages (it needs about 2-3 pages as input). Similar to most of the existing Web data extraction systems, FiVaTech detects the DOM schema with unlabeled types as in Figure 5(a). Also, almost all Web data extraction systems ignore the attached JSON data in the DOM trees of some modern JavaScript Web applications at the client side. These JSON data (if exist) contain (partially or totally) the data to be extracted as well as the schema of these data. Furthermore, these JSON data have a label for each schema type. So, GenDE looks for JSON data corresponding to data rich blocks (blocks of DOM trees that contain data to be extracted) in the Web site, and then uses these JSON data to generate the JSON schema of the Web site.

The proposed system has four main steps as shown in Figure 7. First, it filters out template blocks (navigation, advertisement and other blocks that don't have any data to be extracted) of the input DOM trees and only keep the data rich blocks. Second, it constructs the unlabeled DOM schema using FiVaTech. Third, using the constructed DOM schema s, it looks automatically for a JSON data block (if there) that has a schema s' with common co-basic types from s (i.e., partial or full). Last, the constructed DOM schema is updated by marking the detected co-basic types using the JSON schema. In this subsection, we shall discuss all of these steps in details.

4.1.1 Extract data rich blocks

An important step which enhances the efficiency of generating a DOM schema is identifying/extracting the data rich blocks in the pages DOM trees.

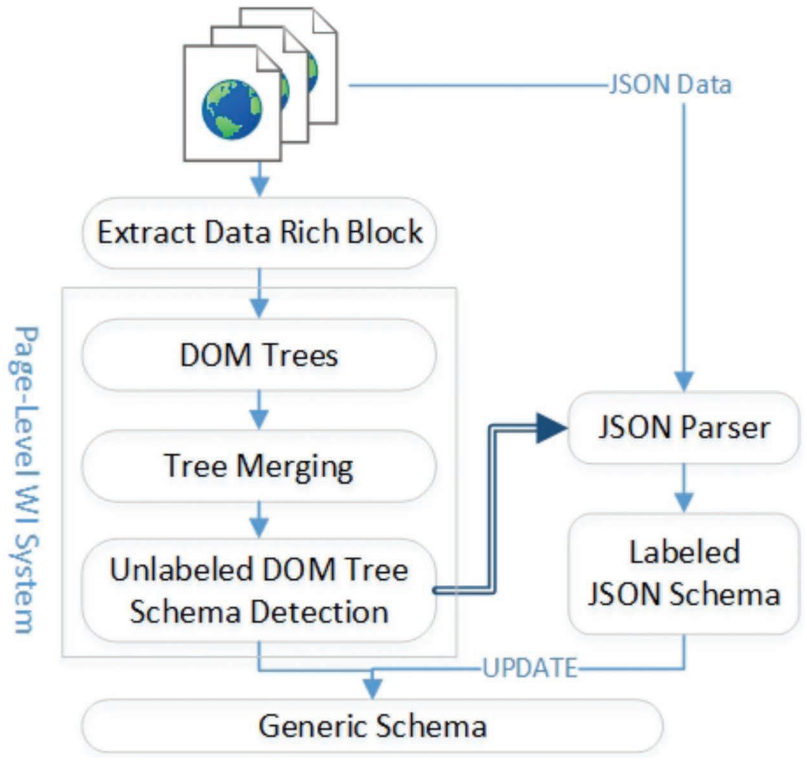


Figure 7 Generic Schema Detection of GenDE.

Our proposed system (similar to [15]) will remove template blocks of the input DOM trees visually by using image comparison. Template data are the blocks of the Web pages that contain advertisements, navigational panels and so on. Data rich blocks are the parts of the Web pages that contain relevant data of interest to the user.

The algorithm uses the block's images rendered by Web browsers. As shown in Figure 8, each tag in the DOM tree has a corresponding image displayed by the browser. The whole image of the page displayed by the browser corresponds to the <body> tag node. The image corresponding to each child node in the DOM tree is presented totally by the browser inside the parent of this node (i.e., images are nested). The dimension and position of a displayed image are provided by the browser via “offsetWidth and offsetHeight” and “offsetLeft and offsetTop”, respectively. The algorithm has two main assumptions/observations:

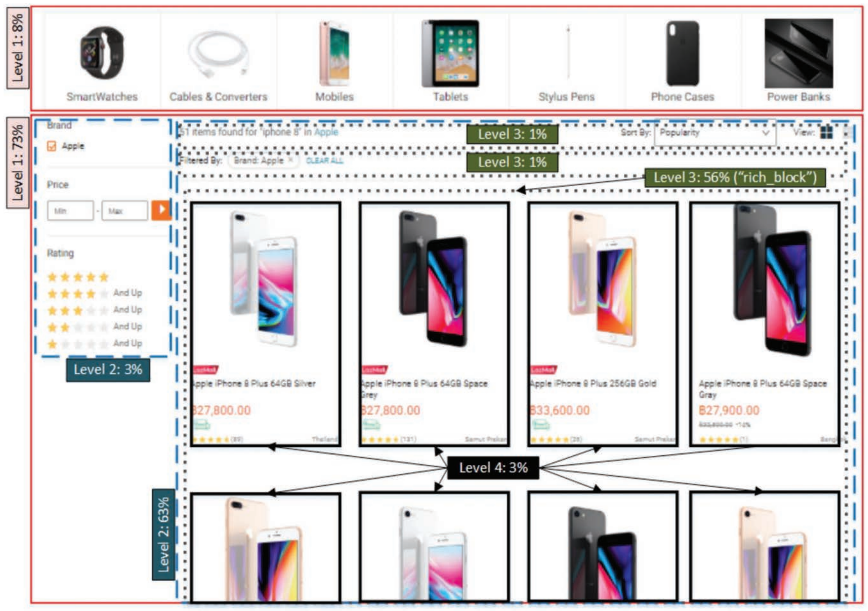


Figure 8 Area Percentages of the pages in Figure 1 at different DOM tree levels.

- Template blocks of the Web pages (in a Web site) are identical (have the same content). Not only the displayed images look the same, but also the tags corresponding to such template blocks are located in the same path of the DOM trees.
- A data rich block often occupies the biggest area in the whole page.

Thus, the algorithm recursively traverses the input DOM tree from the root downward, identifies all child nodes with dimension area percentages greater than a threshold (40%) and labels each of these nodes as a "rich_block". If there is no any child nodes marked as "rich_block", the algorithm stops. Otherwise, the algorithm removes each child node that are not marked as a "rich_block" if it has the same path/content in all pages, and then continuously traverses "rich_block" nodes recursively. For example, as shown in Figure 8, the algorithm starts at the root node which has two child nodes with corresponding images (displayed in red solid rectangles) of area percentages 8% and 73% (Level 1). So, the algorithm will mark the second node (bottom rectangle) as a "rich_block" and remove the first node from all input DOM trees. Similarly, at Level 2, the algorithm will remove the first node (left blue dashed rectangle) and mark the second one (right blue dashed rectangle with percentage (63%) as a "rich_block". Again, at Level 3, the two nodes (top dotted rectangles) will be removed and the one with percentage area 56% is marked as a "rich_block". Finally, it stops at this node because all child nodes have small percentage area (data records in solid rectangles).

4.1.2 DOM schema detection

Given some input DOM trees (data rich blocks) as shown in Figure 7, FiVaTech merges these trees into a single tree (a fixed/variant pattern tree) in which data to be extracted (has a basic type) are identified and marked by an asterisk ("*"), while other HTML tag and text nodes are marked as template. The constructed pattern tree is then used to generate the DOM schema after identifying tuples and lists.

Merging the DOM trees into a single pattern tree has four main steps [5]: peer nodes recognition, matrix alignment, pattern mining and optional node detection. Starting by the root nodes, similar subtrees (peer nodes) on the same level from the input DOM trees are merged and recursively continue until reaching leaf nodes. Peer nodes could be recognized by different algorithms. For example, the tree edit distance algorithm is designed to calculate a matching score for two nodes with the same tag name. If the score is higher than a threshold (0.5), the two trees are considered peer nodes. Although, we use this algorithm for peer nodes recognition, other algorithms are suggested such as using machine learning tools based on visual/content information of the nodes for peer nodes recognition [15].

Child nodes of peer parent nodes are added to a matrix such that all peer child nodes have denoted with the same symbol. The matrix (sequences) alignment algorithm is then used to align child nodes. Many alignment algorithms could be used to solve this problem [21]. The aligned sequence (child nodes) are then mined to discover repeated patterns (set types). The proposed system uses the Tandem repeat mining algorithm for this problem. Finally, optional nodes are identified based on the occurrence vectors. If a set of adjacent nodes have the same occurrence vectors, they will be grouped as optional. By the end of the tree merging step, basic, set and optional types are already detected. The DOM schema is fulfilled by identifying tuples in the pattern tree. This will be done by traversing the pattern tree from the root downward and marking a node as a k-order (if the node is already marked as some data type) or a k-tuple as follows. To avoid too many 1-tuple nodes, nodes with only one child that are not marked as set or optional types are not marked as 1-tuple. Nodes with more than one children are marked as k-order if k children contain a data type. Finally, the DOM schema can then be constructed by removing all tag nodes that have no types.

4.1.3 JSON schema detection

Most Web data extraction approaches use DOM trees of a Web site to detect the schema and extract data from this site. To the best of our knowledge, none of the researchers has focused on automatically extracting JSON data attached in the Web site to be analyzed for schema detection. Only, as mentioned before, [7] has proposed an approach that extracts data from JavaScript Web applications based on some extraction rules and configurations defined manually. Our proposed system will solve the JSON schema detection automatically as follows.

Our approach extracts data rich blocks of the Web pages and generates a DOM schema of these blocks. As emphasized by a doubled-line in Figure 7, the constructed DOM schema shall be used to both decide whether the Web site has some JSON blocks with data to be extracted and (if so) construct the schema of this JSON block by identifying the name/value patterns to be extracted in the found JSON blocks. Given a set A of all basic types in the DOM schema, it will be easy using a JSON parser to calculate the set B which has all basic types δ_i that are co-types for some β_i in A:

$$B = \{\delta_i; \quad \delta_i \text{ is a co-basic type for some } \beta_i \text{ in } A\}$$

The set B will be calculated for each JSON data block attached in the Web page. Now, we have three different alternatives (cases):

(i) There is a JSON data block with a set B such that $|B| = |A|$
(ii) There is a JSON data block with a set B such that $|B| < |A|$
(iii) $B = \phi$ (i.e., B is empty) for all JSON data blocks

The first case (i) means that the Web site has a full JSON schema, the second case (ii) means that the site has a partial JSON schema, while the last case (iii) means that the site has no JSON schema. The JSON schema is constructed (for the first two cases) using the set B. For example, the set $B = \delta_2, \delta_5, \delta_8, \delta_{10}, \delta_{11}, \delta_{13}, \delta_{14}$ for LAZADA Web site which has JSON data shown in Figure 2. At the same time, the set of basic types $A = \beta_3, \beta_5, \beta_6, \beta_9, \beta_{10}, \beta_{11}, \beta_{12}$ for the constructed DOM schema shown in Figure 5(a) (i.e., $|B| = |A| = 7$). So, our proposed algorithm considers LAZADA to have a full JSON schema (shown in Figure 5(b)). The constructed JSON schema is then used to identify the part (node) in the JSON data block that exactly contains the data to be extracted. Usually such JSON blocks contain many other data. The system marks the node which is the direct parent (type) for all instances of the basic types in B as the root of the JSON data to be extracted. For example, the root of the full JSON schema in Figure 5 is the list ($[set]_1$) with label “listItems” in the JSON data block. The constructed JSON schema (partial/full) is then used to update the DOM schema by labeling each type using its corresponding label in the JSON schema. For example, the basic types of the constructed unlabeled DOM schema $\beta_3, \beta_5, \beta_6, \beta_9, \beta_{10}, \beta_{11}$ and β_{12} will take the labels “image”, “name”, “priceShow”, “originalPriceShow”, “discount”, “ratingScore” and “review”, respectively. Many tools (such as Schema Guru[4]) available to generate the JSON schema from JSON data. Practically, we have used some heuristics to fast the process of looking for a JSON data block such as ignoring <script> tags with an attribute “text/html”.

4.2 Wrapper verification

As mentioned before, given a new page from a Web site at a future time t', wrapper verification is the process of tracking any change in the site template/schema in this page. Consequently, it reconstructs the extraction rules (wrappers) if some changes have been detected, and extracts the data from this new page otherwise. To do that, our proposed wrapper verifier uses the previously extracted data, the detected template and the constructed schema of the site at a previous time t. If the site has a full JSON schema, our verifier will extract the attached JSON data and decide whether it follows the previously constructed JSON schema or not. The verification here simply searches for the labels (names) in the name/value pairs of the JSON data from the new page. It starts by searching about the root label that contains all data to be extracted (e.g., “listItems” in the Lazada JSON schema mentioned above). If some of the existing labels (names) are missed in the new JSON data, our verifier would be failed and the unsupervised wrapper induction is called again to generate a new wrapper for the site (see Figure 6). Otherwise, the verifier will straightforwardly extract data (values) from the name/value pairs in the new page. For a type τ which is labeled by a name l, instances of this type will be all values in the embedded JSON data with a name (label) l.

In general for the generated site schema, given a new page as input, a data extractor compares between the input page DOM tree and the constructed schema which has nested sets and options. Comparing schema types from

[4]https://github.com/snowplow/schema-guru

top to down is a challenge as it is very tricky to handle mismatch of tags at internal nodes and to backtrack alternative matches for adjacent optional or set data with the same tag name. In this paper, the data extractor based on the sequence labeling model "Conditional Random Fields" which constructs a distribution over labels, conditioned on observations sequence. The CRF-based data extractor will be used to verify both data semantic and schema structure, and then extract (label) data instances from the input page. Before going to apply the proposed CRF-based data extractor and verifier, to enhance the efficiency of the proposed algorithm, we first check if there is some changes in the template of the Web site using a module called "Template Verifier". The template verifier utilizes an XML validation algorithm to detect whether the page follows the generated XML schema based on the template. If there is some change in the template, the verifier would be failed and our wrapper induction would be called again to generate a wrapper for this site. If the template does not change, the verifier will apply the proposed CRF-based algorithm to verify the schema and extract data from the input page. In this subsection, we shall discuss the two verifiers in details as well as we introduce the CRF sequence labeling model.

4.2.1 Template verifier

Given some pages from a Web site at time t that are used to construct the site schema, the main purpose of the template verifier is to check whether the template of a new result page from this site (at time t') is different from the old template of the pages at time t. Instead of direct comparison of the new page with the template tree, we propose a simple method that uses XML validation to check whether the XML file (transformed from the new page) conforms to the XML schema transformed from the old pages. The template verifier includes three modules: HTML2XML, XML2XSD and XML validation. The first module (HTML2XML) transforms an HTML page into an XML file. The second module (XML2XSD) summarizes an XML schema representation, i.e. XSD (XML Schema Description) from a set of XML pages. The third module checks the validity of the XML page against the XSD specified. The verifier works as follows. It first generates the schema description xsd from the xml files corresponding to the pages at time t. Then, it validates the xml file (corresponding to a new page at t') using the constructed xsd. The benefit here is that all these modules can be done with existing tools. For example, the W3C-defined XML Schema is supported by .NET Framework's Base Class Library, which has a namespace called "System.xml" that provides standards-based support for processing

XML, including reading, writing, schemas searching, transforming and XML validation. However, the issue here is that XSD requires each element at the same level to have distinct names. For HTML tags that are used at the same level, they would be considered as repeated elements even if they contain different subtrees, leading to incorrect schema. Thus, the XML validation module takes the XSD file of the template (at time t) and the XML file of the new page as inputs, and checks whether the XML file is valid with respect to the XSD file. The XML validation module is utilized to only verify the template (not the schema) of a new HTML page, while the rest is handled by the CRF-based verifier.

4.2.2 Conditional random fields

Conditional Random Fields (CRFs) are discriminative graphical models that can model non-independent features. Linear-chain CRF is a special case of this model which is considered the undirected graphical model version of HMM. Given a set of observations $x = x_1, x_2, ..., x_N$ and a set of hidden labels $y = y_1, y_2, ..., y_N$, a linear chain CRF defines the conditional probability $p(y/x)$ as:

$$p(y/x) = \frac{1}{Z} exp(\sum_{t=1}^{N} \sum_{i=1}^{F} \lambda_i f_i(y_{t-1}, y_t, x, t))$$

The first sum runs over each position in the sequence and the second sum runs over each real-valued feature function f_i which is weighted by a parameter λ_i. The feature functions express some characteristic of the empirical distribution that we wish to hold in the model distribution. The factor Z is called a normalization (partition function) scalar which makes the conditional probability valid and it depends on the observation x and the parameters λ. The factor Z is defined as:

$$Z = \sum_y exp(\sum_{t=1}^{N} \sum_{i=1}^{F} \lambda_i f_i(y_{t-1}, y_t, x, t))$$

Given a trained model, finding the most likely label sequence could be measured as follows.

$$\hat{y} = \arg\max \log p(y/x) = \arg\max \sum_{t=1}^{N} \sum_{i=1}^{F} \lambda_i f_i(y_{t-1}, y_t, x, t)$$

This inference could be measured efficiently using a dynamic programming algorithm such as Viterbi algorithm. The parameters λ_i are estimated and learnt from training data D by using gradient ascent. Figure 9 shows the structure of a linear chain CRF. Shaded boxes represent factorization between states (Transition Factors) or between observations and a state (Emission Factors), shaded circles represent observations x (multiple observations for each state), while white circles represent hidden labels y.

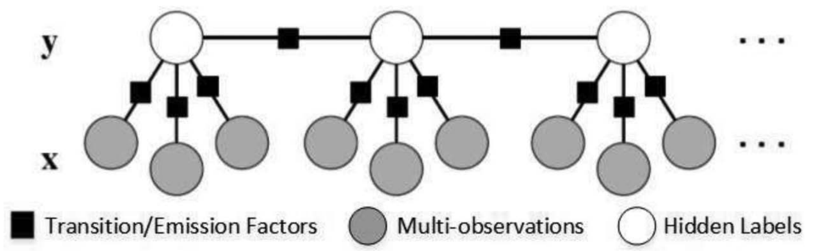


Figure 9 A structure of a linear chain CRF in which each state depends on multiple observations.

4.2.3 CRF-based data extractor

Our proposed CRF-based verifier takes as input data extracted and a schema generated by our WI system, and it aims to verify and extract data from a new test page from the same Web site. Fortunately, the extracted data received are labeled. Although, these labels are anonymous (β_i), our CRF-based algorithm would use them as training data to calculate the feature functions and estimate the CRF model parameters. Sometimes, as discussed before, data labels are updated (with sematic labels) from the JSON schema, if there. Therefore, the observations x to be labeled are the data instances embedded as leaf text/image nodes in the DOM tree of the new page, while the hidden labels are the basic types β_i (anonymous or named) identified in the schema of the Web site. The challenge here is that, data instances of these basic types (labels) are sometimes missed, multi-valued or embedded in the DOM tree using similar template (HTML path). As an example, Figure 10 shows a fictional part of a Web page to be labeled, where the schema of this segment of DOM tree is shown at the top right of the figure. As shown in the figure, data (observations) to be extracted and labeled are embedded as leaf text/image nodes. This means, $x = \{x_1 =$ ”prod-name”$, x_2 = 10, x_3 =$ ”IMG1”$, x_4 =$ ”Name 1”$, x_5 =$ ”\$80”$, x_6 =$ ”\$100”$, x_7 =$ ”-20%”$, x_8 =$ ”IMG2”$, x_9 =$ ”Name 2”$, x_{10} =$ ”\$150”$,x_{11} =$ ”50”$\}$. Based on the schema of this segment, the most likely label sequence for x will be $y = \{y_1 = \beta_2, y_2 = \beta_4, y_3 = \beta_5, y_4 = \beta_6, y_5 = \beta_7, y_6 = \beta_{10}, y_7 = \beta_{11}, y_8 = \beta_5, y_9 = \beta_6, y_{10} = \beta_7, y_{11} = \beta_{12}\}$.

The challenge of applying CRF to label such sequence corresponding to a new Web page is that the sequence length is very long (a whole page), it has missing observations, some observations occurred multiple times, etc. In this subsection, we shall present the details of the proposed CRF-based extractor algorithm and discuss our suggestion to handle this challenge.

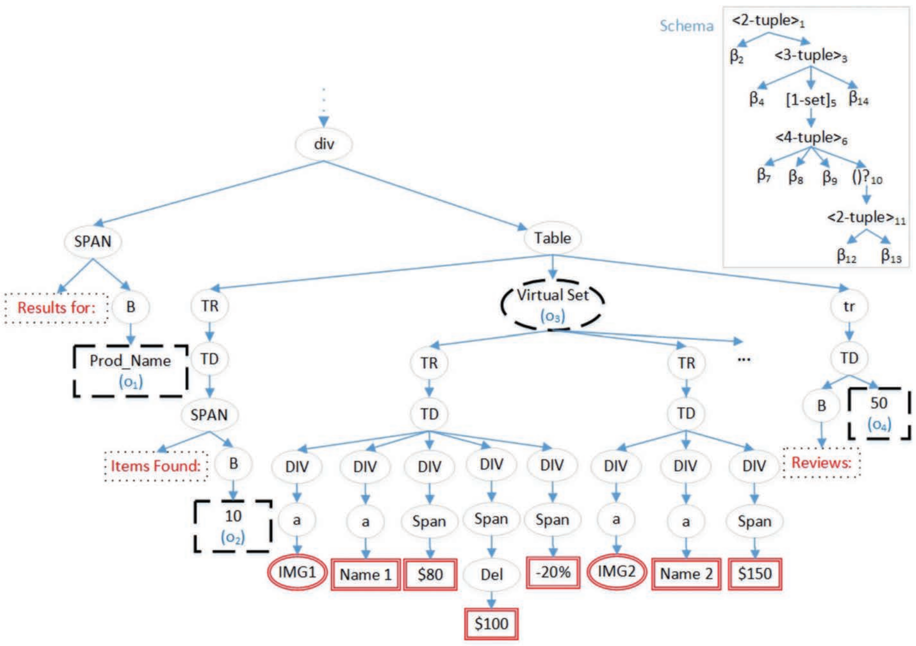


Figure 10 A fictional segment of a Web page and its schema (at the top-right).

4.2.3.1 CRF-based extractor algorithm

The proposed "CRF-Based-Extractor" algorithm takes three arguments as input for a Web site. The first input is a DOM tree of a new page p from the site rooted at node $root$ ($< body >$), the second one is the schema s extracted previously by the unsupervised WI system and finally the training set $D = (x^i, y^i); i = 1, ..., m$ of m sequences, where each sequence vector is labeled by a vector of types from the schema s (D is also provided by the WI system). As mentioned above, it is a challenge to label a big sequence using CRF. So, to handle this problem, our proposed CRF-based extractor algorithm will break down this big sequence into smaller sub-sequences as follows. As shown in Algorithm 1, the algorithm starts by adding and marking a virtual node called "virtual-set" in the DOM tree for each set type in the schema s. This will be done by calling a module "MarkVirtualSetNodes" (step 2), which uses the template of each set type in s (constructed by the unsupervised WI system), identifies all instances of this set type and adds a virtual node (virtual-set) as a parent for all of these set instances in the DOM tree of p. Instances of this set type will be identified based on the node path and the tree-edit distance between each instance subtree and the template of the set type [5]. For example, for the set type ($[]_5$) of the schema at the top right of Figure 10, a virtual-set is added as a parent for the instances of this set type in the DOM tree (a dashed oval in the DOM tree). Finally, in step 3, the module “CRF-BasedSequenceLabeling” is called to recursively break down the observation vector into smaller sequences and then label each sequence as follows.

```
Algorithm 1: CRF-Based-Extractor
  Input :
      $p \rightarrow$ A new input test page
      $s \rightarrow$ A Web site schema
      $D \rightarrow$ A set of sequences and their corresponding labels
1 $root =< Body >$ node of the page $p$
2 MarkVirtualSetNodes($root$, $s$, $D$)
3 CRF-BasedSequenceLabeling($root$, $s$, $D$)
```

As shown in Algorithm 2, the module starts with an empty observation sequence x and an empty set of hidden labels, then it calls the method “PathGuidedObservationLabelMarking”, which is used to get (update) the subsequence x from the DOM tree rooted at “root”. Recursively, the later method traverses the DOM tree starting at the node “root” in a pre-order traversing fashion (steps 6-7 in Algorithm 3). At each node, it checks whether the node is an observation node or not (step 1 in the algorithm). The node is an observation either if it is a text/img node which has a corresponding basic type β in the schema s with the same path as the node or it is a virtual set node (added previously by the method MarkVirtualSetNodes). There may be more than one basic types with a similar path as the node. If so, all of these basic types (a set τ in the algorithm) are considered as candidate types and added to the set of hidden labels Υ (steps 2-3), while the node is added to the observation sequence x (step 4). The proposed algorithm does not aim to label virtual-set nodes, they are just used to simplify and breaking down the sequence into simpler subsequences. After the module “PathGuidedObservationLabelMarking” gets the sequence x at step 3 in Algorithm 2, the CRF model shall be used to label this sequence x by the types from Υ (step 4 in the algorithm). Finally, next subsequences will be obtained (and then labeled) by recursively calling the module "CRF-BasedSequenceLabeling" for each virtual set in the detected sequence x (steps 5-7 in Algorithm 2).

Algorithm 2: CRF-BasedSequenceLabeling

```
  Input :
      $root \rightarrow$ The root node of the new page DOM tree
      $s \rightarrow$ A Web site schema
      $D \rightarrow$ A set of sequences and their corresponding labels
1 Observation $x = \phi$
2 Label set $\Upsilon = \phi$
3 PathGuidedObservationLabelMarking($root$, $s$, $D$, $x$, $\Upsilon$)
4 $\hat{y} = MostLikelyLabel(x, \Upsilon)$ // using CRF model
5 foreach virtual-set node $\upsilon$ in $x$ do
6     CRF-BasedSequenceLabeling($\upsilon$, $s$, $D$)
7 end
```

As an example, the observation sequence of the input page in Figure 10 is divided into two simpler subsequences as follows. The first call of the method PathGuidedObservationLabelMarking on the DOM tree in Figure 10 will assign the four observations (dashed line nodes in Figure 10) as the first subsequence (i.e., $x = \{o_1 =$ "Prod_Name", $o_2 =$ "10", $o_3 =$ "virtual-set", $o_4 =$ "50" $\}$). The set of candidate labels for this subsequence is $\Upsilon = \{S_0, \beta_2, \beta_4, [set]_5, \beta_{14}, S_{end}\}$. The second subsequence after the second traversing will be the doubled line nodes with red color in the subtree rooted at the virtual-set (o_3) in Figure 10. The set of candidate labels for this subsequence is $\Upsilon = \{S_0, \beta_7, \beta_8, \beta_9, \beta_{12}, \beta_{13}, S_{end}\}$, where a start state S_0 and an end state S_{end} are added for each sequence. The algorithm does not consider the leaf nodes that are part of the schema template as observation nodes. For example, the dotted line nodes in Figure 10 are considered as template nodes and they are not considered as observations. For each subsequence, the module CRF-BasedSequenceLabeling (step 4 in Algorithm 2) applies the CRF model to get the most likely labels $\hat{y}$ for the subsequence

x given the candidate labels Υ. The CRF model uses the emission and the transition feature functions that will be discussed in the next subsection. The model uses gradient ascent that maximizes the feature functions to estimate the model parameters.

Algorithm 3: PathGuidedObservationLabelMarking

```
Input :
    $root$ → The root node of the new page DOM tree
    $s$ → A Web site schema
    $D$ → A set of sequences and their corresponding labels
    $x$ → A sequence of observations
    $\Upsilon$ → A sequence of labels
1 if $root$ is an observation node then
2     $\tau$ = set of types in $s$ having same path as $root$. // Candidate labels
3     $\Upsilon = \Upsilon \bigcup \tau$
4     $x = x \bigcup root$
5 end
6 PathGuidedObservationLabelMarking($root.left$, $s$, $D$, $x$, $\Upsilon$)
7 PathGuidedObservationLabelMarking($root.right$, $s$, $D$, $x$, $\Upsilon$)
```

4.2.3.2 Emission and transition feature functions

As mentioned above, a node is considered as an observation node if it is either a leaf text/img node of a basic type with the same path or it is a virtual-set node. For a text/img leaf node, it may has many candidate basic nodes with the same path. The proposed CRF-based algorithm will handle that by both emission feature functions (characteristics of the text/img data instances) and transition feature functions (previous labeled type).

For a basic type β, we use some features to characterize the text strings of the previously extracted instances of β to calculate the emission feature function ($f(\beta, x)$) of an instance x with the type β. These features are alphabetic ratio (LetterDensity), digit ratio (DigitDensity), punctuation ratio (PunDensity), capital-start token ratio (CapitalStartTokenDensity), whether the string is a currency format or not (IsCurrency), whether the string is a URL or not (IsHttpStart) and whether the leaf node is preceded by a template node (IsPrecededByTemplateNode). These features could be described as follows:

- LetterDensity: Density of letters in the string. E.g. LetterDensity(“iPhone 8 256 GB”) =$8/15$.
- DigitDensity: Density of digits in the string. E.g. DigitDensity(“iPhone 8 256 GB”) = 4/15.
- PunDensity: Density of punctuations in the string. E.g. PunDensity(“iPhone 8 256 GB”) = 0.
- CapitalStartTokenDensity: Density of tokens that begins with upper letter. E.g. CapitalStartDensity(“iPhone 8 256 GB”) = 1/4.
- IsCurrency: Whether the string is currency or not. E.g. IsCurrency(“iPhone 8 256 GB”) = 0.
- IsHttpStart: Whether the string starts with “http” or not. E.g. IsHttpStart(“iPhone 8 256 GB”) = 0.
- IsPrecededByTemplateNode: Whether a leaf node is preceded by a template node or not. When traversing the DOM tree, each node comes after a leaf node (such as dotted line nodes in Figure 10) is marked and then be used to enhance the emission function of this node. As an example, IsPrecededByTemplateNode(o_4) $= 1$ for the dashed line leaf node labeled by o_4 in the figure as it has a preceding template leaf (dotted-line) node labeled by “Reviews”.

The mean vector of the basic type training instances ($\mu(\beta)$) in D are used to denote the characters of this type. Finally, the emission function $f(\beta, x)$ is calculated as:

$$f(\beta, x) = \frac{1}{[\epsilon + dist(x, \mu(\beta))]}$$

The distance $dist(x, \mu(\beta))$ is calculated based on the normalized Euclidean distance as:

$$dist(x, \mu(\beta)) = \sqrt{\sum_i \frac{(x_i - \mu_i)^2}{\sigma_i^2 + \epsilon}}$$

Where x denotes the feature vector for a new instance, ($\mu(\beta)$) denotes the mean feature vector of the basic type training instances, σ_i^2 is the variance vector of the features and ϵ (0.0001) is a small value to avoid zero denominator. For the basic types with image instances, simpler boolean feature functions such as ImageClass (that defines a classname of the image), ImageID (that define a specific name of the image), ImageAlt (an alternate text of the image), IsMap(whether the image is a server-side image-map), IsPrecededByTemplateNode (similar to the text nodes), etc. The formula $f(\beta, x)$ is used in the CRF model to calculate the most likely label, but the proposed algorithm adds a restriction to label x by the type β that $f(\beta, x) \geq \varsigma$, where ς is a threshold that is determined experimentally. If $f(\beta, x) < \varsigma$, the extractor would call the un-supervised WI system to re-construct the wrapper.

Finally, the transition function $f(y_{t-1}, y_t, x, t)$ for the two types y_{t-1} and y_t is simply calculated by counting the number of times the type y_t is preceded by the type y_{t-1} ($count(y_{t-1}y_t)$) and the total number of occurrences of the type y_t ($count(y_t)$) in the training set D as:

$$f(y_{t-1}, y_t, x, t) = \frac{count(y_{t-1}y_t)}{count(y_t)}$$

Also, the proposed algorithm decides to label x by the type y_t if the type y_t comes after the preceding type y_{t-1} at least once in the training data, otherwise the WI system is called to re-construct the wrapper.

5 Experiments

Two different experiments are conducted to evaluate the two main addressed problems in the proposed GenDE system. The first experiment evaluates the performance of the system with JavaScript Web sites in which JSON data are embedded to the site pages. The second experiment evaluates the performance of the proposed CRF-based extractor/verifier and compares it with CSP-based extractor/verifier [12], respectively. The details of these experiments are given in the following subsections.

5.1 JSON-based data verification and extraction

In this experiment, we evaluate the performance of the proposed algorithm to identify JSON data embedded in JavaScript Web sites. To the best of our knowledge, no prior benchmark datasets available to evaluate data extraction using the embedded JSON data objects. Therefore, Google search engine is used to randomly collect Web pages from top shopping Web sites. Manually, for each Web site, we identify whether the site pages are embedded by JSON data objects or not. Also, we manually marked the schema of this site as full, partial or not. Some sites such as “Etsy” and “Target” have no JSON data to be extracted although the pages of these sites have some attached JSON blocks, but data in these blocks are embedded for configuration and other setting purposes. Other sites such as “forever21” have added JSON blocks that only contain a part of the data displayed by the browser (to

Table 1 Results of JSON schema identified by the proposed algorithm

Shopping Web Site	Embedded JSON Data	Detected JSON Schema	Valid JSON schema
Lazada	Yes	Full	Full
Walmart	Yes	Full	Full
Etsy	Yes	None	Node
Target	Yes	None	None
Aliexpress	Yes	None	None
Costco	None	None	None
Kohls shoes	Yes	Full	Full
Boohoo	None	None	None
Lululemon	Yes	Full	Full
Athleta	Yes	None	None
American Eagle Outfitters	Yes	Partial	None
Luckybrand	None	Partial	None
Topshop	Yes	Full	Full
Uniqlo	Yes	None	None
forever21	Yes	Partial	Partial
Nasty gal	None	None	None
Allsaints	Yes	Full	Full
Farfetch	Yes	Full	Full
Simmi	Yes	None	None
Butterflytwists	Yes	None	None

be extracted). Finally, other sites have JSON blocks that contain all data (and more) displayed on the site pages (e.g., Lazada, Walmart, Kohls shoes, Lululemon, etc). As shown in Table 1, out of 20 Web sites, only two schema are incorrectly identified as "Partial JSON schema" while they are not (shaded cells in the table). The results show that many JavaScript Web sites (50%) have data to be extracted embedded in the pages on the client side and could be preferably used rather than extracting these data from DOM trees. Also, to compare the proposed algorithm with a similar work [7], we have downloaded the three URLs: https://www.lazada.co.th/shop-led-tv/, https://www.lazada.co.th/shop-monitors/ and https://www.lazada.co.th/shop-kitchen-and-dining/, that are mentioned and used in their experiment. For each one, 5 Web pages are saved: 2 Web pages are used in FiVaTech (our WI system) and the remaining are used for testing. The proposed extractor was able to identifying the JSON block (along with marking the root node in the identified JSON block) that includes the displayed data (data to be extracted) for each test page in each group. Therefore, our system can perfectly extract all data (with labels) without any error. So, not only the proposed system outperforms the algorithm in [7] which defines the extraction rules manually, but also it identifies the schema and extracts data automatically.

5.2 DOM schema based wrapper verification and data extraction

As mentioned above, just two wrapper maintenance approaches are designed to work with page-level Web data extraction systems: WADaR [13] and CSP-based [12] approaches. To compare our approach with the CSP-based verifier [12], we use the same dataset (Rapture). The schema is generated and data are extracted for each Web site using the core unsupervised WI system FiVaTech. We shall use "accuracy", the ratio between the number of correctly extracted pages and the total number of test pages, to measure the performance of the most likely labels resulted by the proposed CRF-based verifier. Since the CRF model parameters are estimated based on the training data which are the instances extracted by the unsupervised WI system, we shall use different number of pages (and so different sizes of training instances) to measure the performance of the sequence labeling algorithm. Also, the accuracy will be measured when either emission feature functions, transition feature functions or both of them are used to model CRF. As shown in Figure 11, the performance increases as the number of instances (number of pages) is increased for the three cases when either emission functions, transition functions, or both are used. There is no any significant changes when 4 or 5 pages are used for the parameters estimation. Also, it is clear that the performance

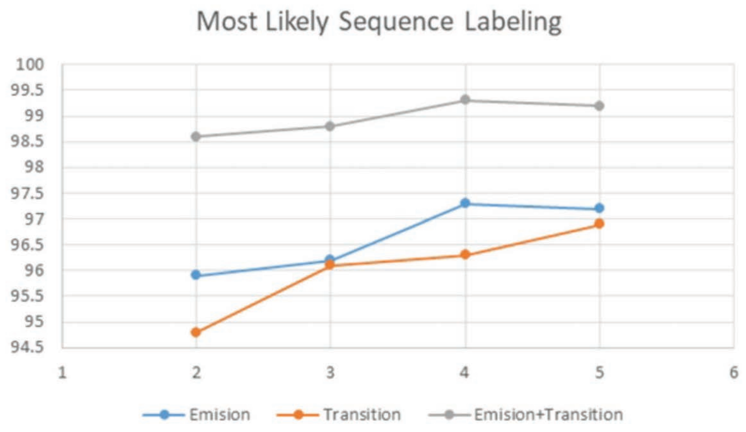


Figure 11 The performance of the CRF sequence labeling algorithm with different number of input pages (2-5) to WI system.

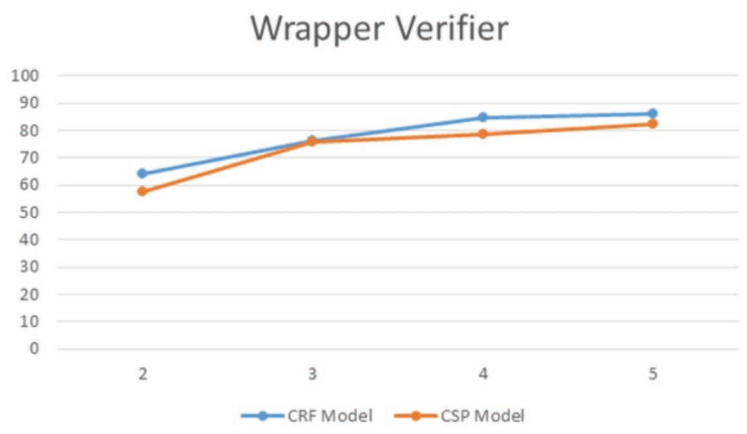


Figure 12 The percentage of pages pass the verifier with different number of input pages (2-5) to WI system.

is the best (with 99.98% accuracy rate average) when both emission and transition functions are used to model the CRF algorithm, while emission feature functions has a better performance (with 96.65% accuracy average) than transition functions (with 96.03 accuracy average).

For the verification problem, we use the same data set collected in [12] which has pages from 9 Web sites (for each site, a set of 20 pages is used). A set of k (2, 3, 4, 5) randomly selected pages are used to train the unsupervised WI for each site. The generated wrapper is tested to see the number of pages from the same site that would pass the verification and the data extraction processes. The process is repeated five times for each site for a given k and the average is taken. The results in Figure 12 show that CRF based verifier outperforms the CSP one. For the two models, using more number of training pages gives better results.

We could not compare our proposed approach with WADaR as the dataset provided by the later approach only includes the data extracted (in Excel format) from 100 different sites (100k Web pages) before and after the wrapper maintenance algorithm has been executed. Different from our proposed approach that verifies Web sites templates/schemas and extracts data from the site pages, WADaR just measures the quality increment of the extracted data after executing the wrapper maintenance algorithm. The dataset used in WADaR is an extension of the dataset used in SWDE [22] (a common Web data extraction benchmark dataset). Therefore, we used SWDE dataset which is collected from 80 Web sites (8 domains: auto, book, camera, job, movie,

Table 2 The performance of the verification algorithm on SWDE dataset

Domain	#Sites	#Pages	#Correctly Verified Pages	Percentage of Correctly Verified Pages
Autos	10	17,923	16,009	89.32%
Books	10	20,000	16,967	84.84%
Cameras	10	5,258	5,008	95.25%
Jobs	10	20,000	15,984	79.92%
Movies	10	20,000	15,763	78.82%
NBA Players	10	4,405	3,979	90.33%
Restaurants	10	20,000	17,479	87.40%
Universities	10	16,705	14,345	85.87%
Total	**80**	**124,291**	**105,534**	**84.91%**

NBA player, restaurant and university) to measure the performance of the verification algorithm. The performance results are shown in Table 2. In this experiment, 2 pages are randomly selected from each Web site (a total of 80 Web sites, 10 sites for each domain) to construct the schema using the WI system. Also, the generated wrapper is tested to see the percentage of pages from the same site that would pass the verification and the data extraction processes. The results show an average of 84.91% of the pages passes the verification and the data extraction processes in a total of 124,291 Web pages.

6 Conclusion and Future Works

Web data extraction from JSON data is an interesting task of great necessity as many Web sites on the Deep Web are attached with these JSON data. As shown in our first experiment, 35% of the randomly selected 20 shopping sites have full JSON schema and 50% of these sites have either partial or full schema. Data extraction from JSON objects provided in currently many modern JavaScript Web sites not only enhances the efficiency and the effectiveness of the Web data extraction approaches, but also it has another important advantage that the extracted data are labeled which is necessary for many intelligent Web applications. We have proposed an algorithm that automatically identifies the part of a JSON block to be extracted and generates its schema. When the Web site has no JSON data, the constructed DOM schema, the template detected and the data extracted previously are used to verify and extract data from a new page at the future. Our proposed data extractor, based on the CRF model, simplifies the sequence labeling problem by breaking down the test page (DOM tree) to be labeled into simpler subsequences. A set of emission and transition feature functions has suggested and used in the proposed CRF-based data extractor GenDE. Our experiment shows an encourage result as it achieves a very high accuracy (99.98% average). Most existing benchmark datasets are designed to evaluate the wrapper induction and schema construction steps. A threat to validity of this data extractor and the other previously proposed extractors is the availability of well formed benchmark datasets. From our point of view, the lack of such big and well formed benchmark datasets is the reason behind not applying deep learning models strongly in this domain. Therefore, researchers in this domain are invited to create and form big and well designed benchmark datasets that are able to perfectly evaluate the two mentioned tasks of wrapper maintenance and data extraction from the Web sites provided or not provided by JSON data.

References

[1] G. M. R. Chang C.-H., Kayed M., S. K. F., A survey of web information extraction systems, IEEE Transactions on Knowledge and Data Engineering 18 (110) (2006) 1411–1428. doi:10.1109/TKDE.2006.152.

[2] H. D. Verberne S., Sappelli M., K. W., Evaluation and analysis of term scoring methods for term extraction, Information Retrieval Journal 19 (5) (2016) 510–545. doi:10.1007/s10791-016-9286-2.

[3] W. Q., N. Y.-K., An ontology-based binary-categorization approach for recognizing multiple-record web documents using a probabilistic retrieval model, Information Retrieval Journal 6 (3-4) (2003) 295–332. doi:10.1023/A:1026024513043.

[4] C. B. Wei X., M. A., Table extraction for answer retrieval, Information Retrieval Journal 9 (5) (2006) 589–611. doi:10.1007/s10791-006-9005-5.

[5] K. M., C. C.-H., Fivatech: Page-level web data extraction from template pages, IEEE Transaction on Knowledge and Data Eng. 22 (2) (2010) 249–263. doi:10.1109/ICDMW.2007.95.

[6] S. H.A., C. R., Tex: an efficient and effective unsupervised web information extractor, Knowledge Based Systems 39 (2013) 109–123. doi:10.1016/j.knosys.2012.10.009.

[7] N. W., P. K., Towards data extraction of dynamic content from javascript web applications, International Conference on Information Networking (ICOIN). doi:10.1109/ICOIN.2018.8343218.

[8] H. D. Meng X., L. C., Schema-guided wrapper maintenance for web data extraction, 5th ACM international workshop on Web information and data management (2003) 1–8. doi:10.1145/956699.956701.

[9] A. M. Raposo J., Pan A., H. J., Automatically maintaining wrappers for semi-structured web sources, Data & Knowledge Engineering 62 (2) (2007) 331–358. doi:10.1016/j.datak.2006.06.006.

[10] M. S. N. Lerman K., K. C. A., Wrapper maintenance: A machine learning approach, Journal of Artificial Intelligence Research 18 (1) (2003) 149–181. doi:10.1613/jair.1145.

[11] L. X. Pek E.-H., L. Y., Web wrapper validation, Proceedings of the 5th International Asia-Pacific Web Conference. doi:10.1007/3-540-36901-5_40.

[12] L. K.-C. Chang C.-H., Lin Y.-L., K. M., Page-level wrapper verification for unsupervised web data extraction, International Conference on Web Information Systems Engineering (WISE) (2013) 454–467. doi:10.1007/978-3-642-41230-1_38.

[13] F. T. Ortona S., Orsi G., B. M., Joint repairs for web wrappers, IEEE 32nd International Conference on Data Engineering (ICDE). doi:10.1109/ICDE.2016.7498320.

[14] B. M. Ortona S., Orsi G., F. T., Wadar: Joint wrapper and data repair, Proceedings of the VLDB Endowment 8 (12). doi:10.14778/2824032.2824120.

[15] C. C.-H. Chang C.-H., K. M., Fivatech2: A supervised approach to role differentiation for web data extraction from template pages, 26 th annual conference of the Japanese Society for Artifical Intelligence, Special Session on Web Intelligence & Data Mining (2012) 1–9.

[16] P. J. Eric F.-L., Yanzhang H., P. R., Conditional random fields in speech, audio, and language processing, Proceedings of the IEEE 101 (5). doi:10.1109/JPROC.2013.2248112.

[17] L. X., B. D., Extracting addresses from news reports using conditional random fields, 15 th IEEE International Conference on Machine Learning and Applications. doi:10.1109/ICMLA.2016.0141.

[18] W. J.-R. Z. B. Zhu J., Nie Z., M. W.-Y., 2d conditional random fields for web information extraction, Proceedings of the 22 nd international conference on Machine learning (ICML) (2015) 1044–1051. doi:10.1145/1102351.1102483.

[19] X. R. Liu R., G. K., A crf-based approach for web object extraction, 3 rd International Conference on Computer Science and Information Technology. doi:10.1109/ICCSIT.2010.5563787.

[20] K. N., Wrapper verification, World Wide Web Journal 3 (2) (2000) 79–94. doi:10.1023/A:101922961.

[21] K. M., Peer matrix alignment: a new algorithm, Pacific-Asia Conference on Knowledge Discovery and Data Mining (ICDM) (2012) 268–279. doi:10.1007/978-3-642-30220-6_23.

[22] P. Y. Hao Q., Cai R., Z. L., From one tree to a forest: a unified solution for structured web data extraction, SIGIR, Beijing, China (2011) 775–784. doi:10.1.1.229.2837.

Biographies

Mohammed Kayed received his M.Sc. degree in Computer Science from Minia University, Minia, Egypt, in 2002 and the Ph.D. degree in Computer Science from Beni-Suef University, Beni-Suef, Egypt, in 2007. From 2005 to 2006, he was a Research&Teaching Assistant in Department of Computer Science and Information Engineering at the National Central University, Taiwan. From 2008 to 2015, he was an Assistant Professor, IT Department, College of Applied Science, Sultanate of Oman. He is currently an Associate Professor and Head of Computer Science Department, Faculty of Computer and Artificial Intelligence, Beni-Suef University, Egypt. He is the author of more than 25 articles. His research interests include Web mining, Opinion Mining, Information Extraction and Information Retrieval.

Khaled Shaalan is a full professor of Computer Science/Artificial Intelligence at the British University in Dubai (BUiD), UAE. He is an Honorary Fellow at the School of Informatics, University of Edinburgh (UoE), UK. Over the last two decades, Prof Khaled has been contributing to a wide range of research topics in AI, Arabic NLP, Knowledge management, health informatics, and educational technology. Prof Khaled has published 200+ referred publications. Prof Khaled's research work is cited extensively worldwide and the impact of his research using GoogleScholar's H-index metric is 35+. Prof Khaled has been actively and extensively supporting the local and international academic community. He acts as the chair of international Conferences, journals & books editor, keynote speaker, external member of promotions committees, among others.