*International Journal of Computer Applications (0975 – 8887)*
*Volume 107 – No. 1, December 2014*

# A Classifier for Schema Types Generated by Web Data Extraction Systems

Mohammed Kayed
Faculty of Science
Beni-Suef University, Egypt

Awny Sayed
Faculty of Science,
Minia University, Egypt

Marwa Hashem
Faculty of Science,
Beni-Suef University, Egypt

## ABSTRACT

Generating Web site schema is a core step for value-added services on the web such as comparative shopping and information integration systems. Several approaches have been developed to detect this schema. For a real web site, due to the complexity of the site schema, post process of this schema such as labeling the schema types, comparing among different schema types and generating an extractor to extract instances of a schema type is a challenge. In this paper, a new tree structured called schema-type semantic model is proposed as a classifier for a schema type. Given some instances of a schema type, HTML tags contents, DOM trees structural information and visual information of these instances are exploited for the classifier construction. Using multivariate normal distribution, the classifier can be used to compare between two different schema types; i.e., the classifier can be used for schema mapping which is a core step of information integration. Also, the suggested classifier can be used to detect and extract instances of a schema type; i.e., it can be used as an extractor for web data extraction systems. Furthermore, the classifier can be used to improve the performance of the schema generated by web data extraction systems; i.e., the classifier can be used to get, as much as possible, a perfect schema. The experiments show an encourage result with the schemas of the test web sites (a data set of 40 web sites).

## General Terms

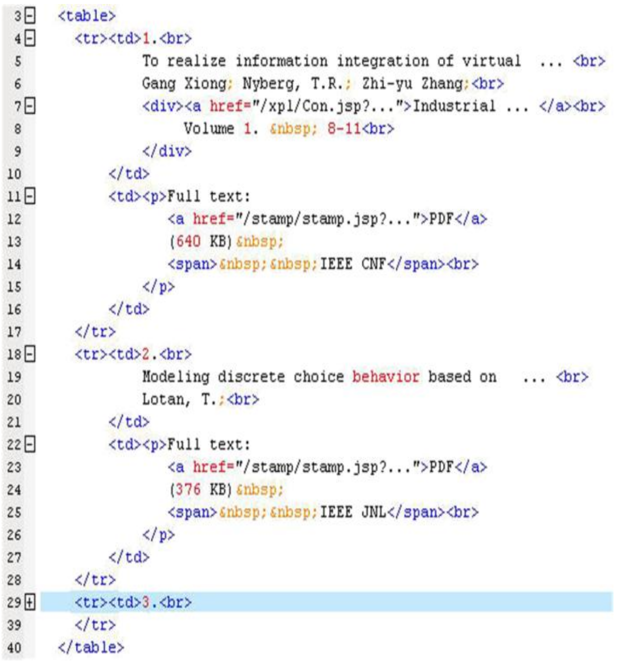
Information Extraction, Schema-Type Classifier

## Keywords

Schema Mapping, Schema Type Classifier, Schema Filtration, Web Data Extraction

## 1. INTRODUCTION

The explosive growth and popularity of the World Wide Web has resulted in a huge amount of information sources on the Internet. However, due to the heterogeneity and the lack of structure of web information sources, access to this huge collection of information has been limited to browsing and searching. This cost of browsing is becoming noticeable with the Deep Web (Invisible Web), which contains magnitudes more and valuable information than the Surface Web. Web pages in the Deep Web share the same template since they are encoded in a consistent manner across all pages. In other words, these pages are generated with a predefined template by plugging a data instance. Embedding a data instance *x* into the template T is an encoding process, λ(T; *x*), which generates HTML pages. Unsupervised wrapper induction is basically a reverse engineering of the page generation model, which induces the template and schema from a set of given pages and extracts the embedded data. Figure 1 gives an example of such web pages in which three data records are presented. Each data record is embedded in an HTML <tr> tag. The schema of a data instance in a web page can be defined as follows.



```
<table>
  <tr><td>1.<br>
          To realize information integration of virtual  ... <br>
          Gang Xiong; Nyberg, T.R.; Zhi-yu Zhang;<br>
          <div><a href="/xpl/Con.jsp?...">Industrial ... </a><br>
             Volume 1. &nbsp; 8-11<br>
          </div>
      </td>
      <td><p>Full text:
            <a href="/stamp/stamp.jsp?...">PDF</a>
            (640 KB) &nbsp;
            <span>&nbsp;&nbsp;IEEE CNF</span><br>
          </p>
      </td>
  </tr>
  <tr><td>2.<br>
          Modeling discrete choice behavior based on   ... <br>
          Lotan, T.;<br>
      </td>
      <td><p>Full text:
            <a href="/stamp/stamp.jsp?...">PDF</a>
            (376 KB) &nbsp;
            <span>&nbsp;&nbsp;IEEE JNL</span><br>
          </p>
      </td>
  </tr>
  <tr><td>3.<br>
  </tr>
</table>
```

**Fig. 1: Example of an HTML page.**

**Definition (Structured Data):** A data schema can be of the following types [1]:

1. A basic type $\beta$ represents a string of tokens, where a token is some basic units of text.

2. If $\tau_1, \tau_2, \ldots, \tau_k$ are types, then their ordered list $\langle \tau_1, \tau_2, \ldots, \tau_k \rangle$ also forms a type $\tau$. The type $\tau$ is constructed from the types $\tau_1, \tau_2, \ldots, \tau_k$ using a type constructor of order *k*. An instance of the *k*-order $\tau$ is of the form $\langle x_1, x_2, \ldots, x_k \rangle$, where $x_1, x_2, \ldots, x_k$ are instances of types $\tau_1, \tau_2, \ldots, \tau_k$, respectively. The type $\tau$ is called:

   a. A tuple, denoted by $\langle k - tuple \rangle_\tau$, if the cardinality (the number of instances) is 1 for every instantiation.

   b. An option, denoted by $(k)?_\tau$, if the cardinality is either 0 or 1 for every instantiation.

   c. A set, denoted by $\{k - set\}_\tau$, if the cardinality is greater than 1 for some instantiation.

   d. A disjunction, denoted by $(\tau_1 | \tau_2 | \ldots | \tau_k)_\tau$, if all $\tau_i$ $(i = 1, \ldots, k)$ are options and the cardinality sum of the $k$ options $(\tau_1 - \tau_k)$ equals 1 for every instantiation of $\tau$.

Figure 2 shows the schema of the web page in Figure 1, which presents a list ($\{\}_2$) of publications; each one consists of a 4-tuple ($<>_3$) and an optional 2-tuple ($()?_{10}$). The 4-tuple type includes five basic types (4-9), where the last two are optional. The optional tuple $()_{10}$ has two basic types (11-12). Each type in the schema of a web site has a set of instances in each web page in the site. For example, Figure 3 shows two instances of the set type $\{\}_2$. As shown in Figures 1-3, template, schema, and data instances are tree structured.

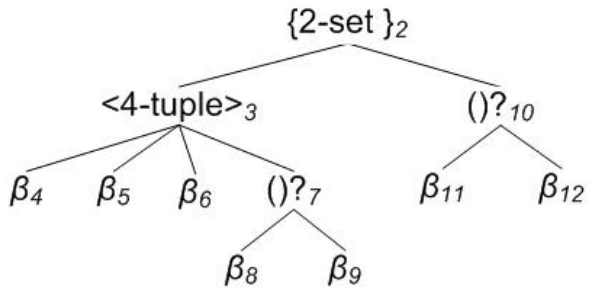


**Fig. 2: The schema of the HTML page in Fig. 1.**

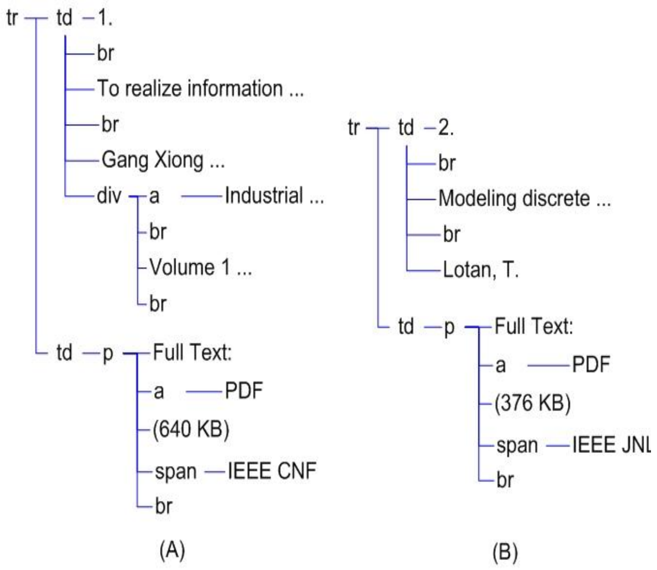


**Fig. 3: Two instances (A and B) of the schema set type $\{\}_2$ in Fig. 2.**

Given some instances of a schema type τ, this paper is aimed to construct a schema-type semantic model that can be used as a classifier for the type τ. HTML tags contents, DOM trees structural information, and visual information of the instances are used for the classifier construction. The constructed classifier has three main contributions. First, it can be used to compare between two different schema types (i.e., schema matching). Second, the classifier of the type τ can be used to decide whether a data value in some web page is an instance of this type (i.e., as an extractor for web data extraction systems). Third, it can be used to improve the performance of the schema generated by web data extraction systems as we shall discuss in details later.

The paper is organized as follows. Section 2 reviews the related works. Section 3 provides the details of the proposed schema-type semantic classifier. Section 4 gives an algorithm to improve the performance of the schema generated by web data extraction systems. The experiments and the conclusions are presented in section 5 and section 6, respectively.

## 2. RELATED WORKS

In the last few years, web data extraction has been a hot topic. Many approaches have been developed with different task domain, automation degree, and techniques [2]. Many of the developed approaches aim to detect the schema of a web site which can be used with the generated wrapper for data extraction. Examples of these wrapper induction systems are EXLAG [1], FiVaTech [3], RoadRunner [4], Dela [5], DEPTA [6], ViPER [7], and others [8-10]. FiVaTech, EXLAG and RoadRunner are designed to solve the page-level extraction task, while DeLa, DEPTA, and ViPER are designed for the record-level extraction task. In this paper, we use the schemas/instances detected by FiVaTech. Post process of the generated schema such as labeling the schema types, comparing among different schema types and generating an extraction module to extract instances of a schema type is an important step as this schema is a core for value-added services on the Web.

The proposed classifier is aimed to "Match" schema types, which produces a mapping between elements of the schema that correspond semantically to each other [11-12]. So, this paper is very relevant to "schema matching" which is a basic problem in many database application domains, such as data integration, E-business, data warehousing, and semantic query processing. Database integration is a process with multiple steps, leading from the identification of the databases to the testing of the integrated system. The central step of the database integration process is the identification of those elements in the schema of the databases that match each other. This step is termed schema matching. Schema matching methods are primarily categorized by their use of schema-level or instance-level information, although many methods use both types of information [13].

- ***Schema-level techniques:***

Three types of information may be used in the schema-level techniques: constraint information, linguistic information and structural information. Examples of constraint information are data type constraints, optionality constraints and uniqueness constraints of attributes [14-15]. Linguistic information may be used by measuring character string similarity [16], or by using externally supplied dictionaries, thesauri or lexical databases [17-18] such as WordNet [19] or CyC [20]. Linguistics based techniques are limited to problems where linguistic information are available. Example of structural information is the relationships between database elements such as relationship-types between entity-types or foreign-key dependencies between tables. The use of structural information for identifying matching database elements may be limited to local structures, where relationships only between directly connected database elements are considered, or it may encompass global structures, where the overall structure of the database is considered.

- ***Instance-level techniques:***

In addition to or as an alternative to schema-level techniques, schema instances could be used for schema matching. For example, a schema-level matcher may be used to match entity types and an instance-level matcher may subsequently be used to match attributes [21]. To match semantically similar attributes, instance information such as attribute value distributions and term frequencies may be used. For example, when two table attributes contain the same distribution of values, then the columns are argued to be similar in meaning. Machine learning techniques such as neural networks [22] and Bayesian learners [21] among others can establish characteristic features of an attribute or column which can then be compared to others.

Various approaches and techniques have been proposed for detecting schema-level correspondences across heterogeneous databases. First, some of them apply linguistic techniques [23], to measure the similarity between the names of schema elements. These approaches are therefore difficult to apply in many legacy systems, where schema elements are not well named using standard terms. Second, some other approaches use heuristic formulae to measure the similarity between schema elements, based on the names and structures of the schema elements [24]. Third, another approach computes the similarity between the text descriptions of schema elements (in design documents) using similarity measures developed in the information retrieval field [25]. The major difficulty with this approach is that the design documents are often out of date, inaccurate, or even not available in many legacy systems. Fourth, other approaches have used cluster analysis techniques to cluster schema elements based on the meta-data (e.g., name, schematic specification, and summary statistics) about the schema elements [26]. However, due to various problems associated with the features for such cluster analysis [27], users must carefully evaluate the results generated by the cluster analysis techniques. In this paper, cluster analysis techniques are used to cluster the similar schema elements.

In the absence of schema element name as the case for the schema detected by web data extraction systems, data instance is considered an important source of semantic information for schema matching. Many characterizations can be exploited for instance-based schema matching. For example, keywords and themes extracted based on the relative frequencies of words and combinations of words, etc. Also, a constraint-based characterization, such as numerical value ranges and averages or character patterns can be applied. For instance, this may allow recognizing phone numbers, zip codes, geographical names, addresses, ISBNs, SSNs, date entries, or money-related entries (e.g., based on currency symbols). Finally, statistical features such as alphabetic ratio (Letter Density), digit ratio (Digit Density), punctuation ratio (Pun Density), and tokenize the strings via capital-start token ratio (Capital Start Token Density) and numerical token ratio, etc, can be used as in [28]. Also, kushmerick [29] introduced RAPTURE which uses 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 the similarities between data observed by the wrapper and that expected.

## 3. THE PROPOSED CLASSIFIER

Web data extraction approaches aim to detect the schema of a deep web site and extract instances of each schema type from some given web pages. Given this extracted schema and the instances of a schema type $\tau$, we suggest a classifier called schema-type semantic model for the schema type $\tau$. The suggested classifier can be used to compare between the schema type $\tau$ and an instance value. So, it could be used as an extractor to extract instances of the type $\tau$ from other web pages. Also, the classifier can be used to compare among different schema types. So, it could be used for schema mapping. Furthermore, the classifier can be used to compare among different instances. In this section, the details of using the instances $x_i$ ($i$=1, …, $k$) to construct the suggested classifier are discussed in details.

If $\tau$ is a basic type ($\tau = \beta$) of $k$ text node instances, i.e., each instance $x_i$ of type $\tau$ is a leaf text node with a text value, then the function $Semantic$ for a basic-type $\beta$ is defined as an $N$-tuple as follows:

$$Semantic(\beta) = \langle a_1, a_2, \dots, a_N \rangle,$$

$a_i = \frac{\sum_{j=1}^{k} f_i(x_j)}{k}$ is the average of the feature values $f_i$ for all of the instances $x_j$ of the basic type $\beta$ which are calculated as shown in Table 1.

The semantic function that is calculated based on statistical features (syntactical/NLP features may also be used in the future) can be used to measure the similarity between the basic type $\beta$ and an instance $x$ using the *multivariate normal* distribution as follows.

$$\text{Basic\_}Sim(x, \beta) = exp\left(-\sum_{i=1}^{N} \frac{(f_i(x) - a_i)^2}{\sigma_i^2}\right)$$

where $f_i(x)$; $i$ =1, …, $N$, are the feature values of the instance $x$, and $\sigma_i$ is the standard deviation of the feature values $f_i$ for all of the instances of the basic type $\beta$. The instance value $x$ is considered similar to the basic type $\beta$ if the value $Basic\_Sim(x, \beta)$ is greater than a threshold which is selected based on the instances extracted by FiVaTech.

**Table 1: List of the features used to calculate the semantic of a basic type.**

| Feature | Name | Description |
|---|---|---|
| $f_1(x)$ | Letter Density | The alphabetic ratio in the instance $x$. |
| $f_2(x)$ | Digit Density | The digit ratio in the instance $x$. |
| $f_3(x)$ | Pun Density | The punctuation ratio in the instance $x$. |
| $f_4(x)$ | Capital Start Token Density | The ratio of the capital-start tokens in $x$. |
| $f_5(x)$ | Is Upper Case | Boolean: true if the instance text value $x$ is upper case, false otherwise. |
| $f_6(x)$ | Is Http Start | Boolean: true if the instance $x$ is a URL, false otherwise. |

If τ is a non-basic (tuple/set) type with tree-structured instances $x_i$ ($i$=1, …., $k$), we propose a classifier of the type τ that exploits structural information, HTML tags contents, visual information, and data semantics of the instances of τ. The proposed classifier for the non-basic type τ includes two parts (phases): semantic-based classifier and statistical-based classifier. The formal one, semantic-based, is a tree structure embedded with semantic data of the text nodes. The later classifier is a $K$-tuple which exploits both visual and HTML tags contents information of the instances of the type τ. An instance $x$ is similar to the non-basic type τ, only if $x$ is similar to both of the two proposed classifier parts. The two classifier parts will be discussed in the following subsections.

### 3.1 Semantic-Based Classifier

The semantic-based classifier of the non-basic type τ (*Semantic-Classifier*(τ)) is constructed by replacing all basic type nodes $\beta_i$ by $Semantic(\beta_i)$ in the schema tree corresponds to the type τ, where $Semantic(\beta_i)$ is calculated as defined above. As an example, Figure 4 is the semantic-based classifier of the set type $\{\}_2$ in the schema given in Figure 2. Every basic type node $\beta$ in the schema tree is replaced by its semantic ($Semantic(\beta)$) to construct the classifier. The semantic-based classifier can also be defined for an instance subtree $x$ (*Semantic-Classifier*($x$)) as follows. *Semantic-Classifier*($x$) is the same subtree as $x$ such that every text node $x_i$ in the instance subtree $x$ is replaced by its semantic *Semantic*($x_i$). In summary, we can notice that *Semantic-Classifier*($\tau$) is carrying both structural and semantic information about the type $\tau$, where these semantic information are accumulated by using the instances of the type τ.

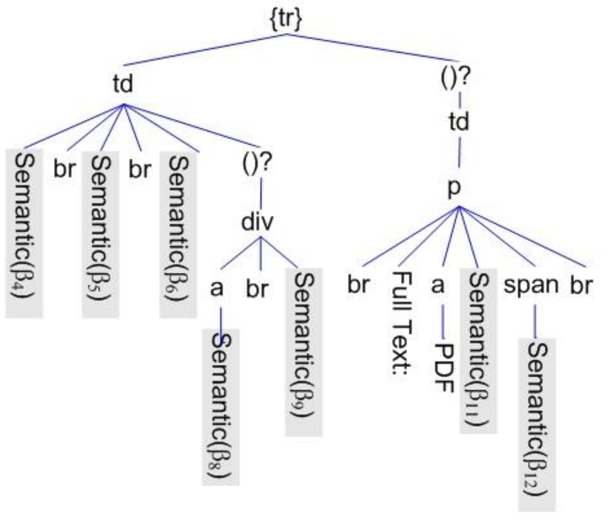


**Fig. 4: the semantic-based classifier of the set type {}2.**

Now, the tree-edit distance is used to measure the similarity between two semantic-based classifiers $C_1$ and $C_2$. We use paths instead of individual nodes for subtrees matching. Let $p_1$ and $p_2$ are two paths in $C_1$, and $q_1$ and $q_2$ are two paths in $C_2$, we define the matching between the two trees $C_1$ and $C_2$ as a mapping $M$ such that for every two pairs $(p_1, q_1) \in M$ and $(p_2, q_2) \in M$, the following two conditions are satisfied:

(1) $p_1 = p_2$ ***iff*** $q_1 = q_2$;

(2) $C_1[p_1]$ is on the left of $C_1[p_2]$ ***iff*** $C_2[q_1]$ is on the left of $C_2[q_2]$;

The first condition requires that each path can appear only once in the mapping, while the second condition enforces the order preservation between the paths in the tree. A maximum matching is a matching with the maximum number of pairs. Finally, similarity between the two classifiers $C_1$ and $C_2$ is calculated as:

$$Semantic\_Sim(C_1, C_2) = \frac{|M|}{Avg\,(n_1,n_2)};$$

where $n_1$ and $n_2$ are the total number of different paths in $C_1$ and $C_2$, respectively, and $|M|$ is the size of the matching pairs (maximum matching) in $M$. The two trees $C_1$ and $C_2$ are similar if their similarity is greater than a threshold.

The two paths $p$ ($a_1$/… / $a_m$) and $q$ ($b_1$/… / $b_m$) are matched (i.e., $(p, q) \in M$)) if there is a one-one corresponding mapping between every two nodes $a_i$ and $b_i$ in the two paths $p$ and $q$, respectively (see Figure 5). The node $a_i$ has a mapping with the node $b_i$ (at the same level $i$) when one of the following cases is satisfied:

- The two nodes $a_i$ and $b_i$ are instances of some basic types and they are similar, i.e., Sim ($a_i$, $b_i$) is greater than a threshold.
- The two nodes $a_i$ and $b_i$ are leaf (text/img) nodes, but they are part of the template of the same value. For example, the text node "Full Text" in Figure 3 is an example of such nodes.
- The two nodes $a_i$ and $b_i$ correspond to a same HTML tag.

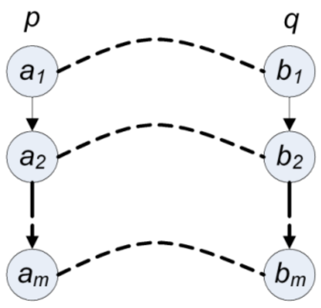


**Fig. 5: A mapping between the two paths p and q.**

## 3.2 Statistical-Based Classifier

We define a statistical-based classifier of the non-basic type τ as a $K$-tuple which exploits both visual and HTML tags contents information of the instances (subtrees) of the type τ:

$$Statistical\_Classifier(\tau) = \langle b_1, b_2, \dots, b_K \rangle;$$

$b_i = \frac{\sum_{j=1}^{m} g_i(x_j)}{m}$ is the average of the statistical feature values $g_i$ for all of the $m$ instances $x_j$ of the type τ which are calculated as shown in Table 2.

The statistical classifier can be used to measure the similarity between the type τ and an instance $x$ using the *multivariate normal distribution* as follows.

$$Statistical\_Sim(x, \tau) = exp\left(-\sum_{i=1}^{K} \frac{(g_i(x) - b_i)^2}{s_i^2}\right)$$

**Table2: List of the features used to calculate the semantic of a basic type.**

| Feature | Name | Description |
|---|---|---|
| $g_1(x)$ | Width Percentage | The percentage of the width of the image corresponds to the instance *x* to the page width. |
| $g_2(x)$ | Height Percentage | The percentage of the height of the image corresponds to the instance *x* to the page height. |
| $g_3(x)$ | Area Percentage | The percentage of the area of the image corresponds to the instance *x* to the whole page area. |
| $g_4(x)$ | Template Percentage | The percentage of the template text nodes in the tree *x*. |
| $g_5(x)$ | Text Nodes Percentage | The percentage of the text nodes in the tree *x*. |
| $g_6(x)$ | Leaves Percentage | The percentage of the leaf nodes in the tree *x*. |
| $g_7(x)$ | Decoration Percentage | The percentage of the decorative tag nodes (that emphasize the text nodes) in the tree *x*. |

where $g_i(x)$; $i$ =1, …, $N$, are the feature values of the instance $x$ (shown in Table 2), and $s_i$ is the standard deviation of the feature values $g_i$ for all of the instances of τ. The instance value $x$ is considered similar to the type τ if the value $Statistical\_Sim(x, \tau)$ is greater than a threshold.

The two formulas *Semantic_Sim* and *Statistical_Sim* can be used now to measure the similarity among different schema types, among different instances, or between a schema type and an instance. The former formula measures both structural and semantic similarities, while the later one measures both content and visual similarities.

## 4. CASE STUDY: SCHEMA FILTRATION

In this section, we use the proposed classifier to post-process of the schema generated by web data extraction systems. We give an algorithm that tries to filter out the schema by using the proposed classifier. The first subsection defines the schema filtration problem. The second subsection illustrates the problem by giving a simple example. Finally, the filtration algorithm is discussed in the third subsection.

### 4.1 Problem Definition

Let $Instance(\tau_1) = \{x_1, x_2, \dots, x_{k_1}\}$ and $\mathrm{Instance}(\tau_2) = \{y_1, y_2, \dots, y_{k_2}\}$ are the instances of the schema types $\tau_1$ and $\tau_2$, respectively. We give the following definitions.

**Definition (Incomplete Schema Type):** *The schema type $\tau_1$ is called incomplete if there is some value x such that $Similarity(\tau_1, x)$ is greater than a threshold, while x is identified as an instance of a different type $\tau_2$; i.e., $\tau_1$ is an incomplete type if there is some value x such that x is similar to the instances of the type $\tau_1$, while $x \notin Instance(\tau_1)$.*

**Definition (Incorrect Schema Type):** *The schema type $\tau$ is called incorrect if there is some instance $x_i \in Instance(\tau)$, where $Similarity(\tau, x)$ is less than a threshold.*

**Definition (Web Site Schema Filtration):** *Given a schema tree, schema filtration is the process of* identifying *incomplete and incorrect schema types, and then handling them to get a perfect/correct (as much as possible) schema.*

In the definitions above, the function $Similarity(\tau, x)$ is used to decide that $x$ is an instance of the type $\tau$ as follows. If $\tau$ is a basic type, then $x$ is an instance of $\tau$ when $Similarity(\tau, x) = basic\_Sim(\tau, x)$ is greater than a threshold. If $\tau$ is a non-basic type, then $x$ is an instance of $\tau$ when each of the two values $Centroid\_Sim(\tau, x)$ and $Statistical\_Sim(\tau, x)$ is greater than a threshold.

### 4.2 An Illustrative Example

In this subsection, we give a simple example to demonstrate the definitions defined above. Figure 6(b) gives the schema identified by FiVaTech for a part of a web page in Figure 6(a), where Figure 6(c) shows the DOM tree of this part of the web page. As shown in the figure, the three instances $i_1$, $i_2$, and $i_3$ have the same structure, so FiVaTech has identified all of them as instances of a same type $\{\}_2$. Similarly, $i_5$–$i_{14}$ are instances of $\{\}_6$. The tuple $<>_4$ has only one instance ($i_4$) which displays the current page in a STRONG html tag. Although all the instances $i_1$-$i_3$ and $i_5$-$i_{14}$ have the same structure (each one displays a link to a web page), semantically, we have two types of data in these instances: link to a particular page and link to next/previous pages. As we defined above, the type $\{\}_2$ is incorrect, because not all of its instances are similar (e.g., $i_3$ is not similar to the two other instances $i_1$ and $i_2$). Similarly, the type $\{\}_6$ is incorrect. Also, the set type $\{\}_6$ gives an example of an incomplete type because the instance $i_3$ is similar to the instances $i_5$-$i_{12}$ although they are identified as instances of different types. The challenge here is how we can filter out such schema by handling incorrect and incomplete schema types.

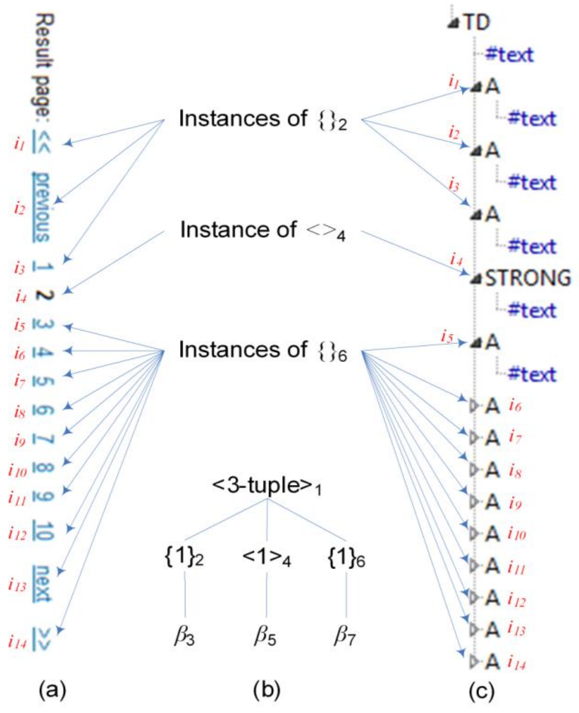


**Fig. 6: (a) A part of a web page, (b) its schema, and (c) its DOM tree.**

### 4.3 Schema Filtration Algorithm

In this subsection, we discuss the details of the proposed algorithm for schema filtration. Experimentally, as clear in the simple example given above in the previous section, incorrect and incomplete schema types almost are set types. For example, both of the two set types {}2 and {}6 are incorrect and incomplete. Also, incomplete non-set types appear with consecutive tuple/basic types. Therefore, the suggested filtration algorithm includes two main steps in order:

**Step 1:** Set-type filtration

**Step 2:** Consecutive non-set types filtration

In the first step, Step I, the proposed algorithm traverses the schema tree and handles incorrect and incomplete types among set-type nodes in the schema as shown in Figure 7. An important notice here is that incorrect schema types should be handled before incomplete types. If we do the converse, we may get an incorrect semantic function $Semantic(\beta)$ for some basic type $\beta$ (i.e., it does not correctly reflect the semantic of the type $\beta$), so we could not correctly measure the similarity between $\beta$ and another type/instance. Furthermore, a semantic-based classifier that has incorrect schema types could not be used perfectly to measure the similarity among non-basic types. For example, in Figure 6, before handling the incorrect schema type $\{\}_2$, the function $Semantic(\beta_3)$ (*Semantic-Classifier*($\{\}_2$)) will be calculated based on the instance values "<<", "previous", and "1". So, the function will not semantically reflect the instance data because they are not semantically similar. Therefore, the proposed algorithm will handle incorrect types before handling of incomplete types. In the second step, Step II, we traverse the schema tree and combine similar consecutive non-set types as a new set type. The similarity is measured based on the formulas defined in Section 4.1. We shall discuss the two steps in details below.

```
Algorithm Set_Type_Schema_Filtration (n, ST, Σ, patTree)
1. for each child c in n
2.    Set_Type_Schema_Filtration (c, ST, Σ, patTree)
3. endfor
4. if (c is a set-type) then
5.    handleIncorrecteType(c, ST, Σ, patTree)
6.    handleIncompleteType(c, ST, Σ, patTree)
7. endif
```

**Fig. 7: The proposed set-type filtration algorithm.**

As shown in Figure 7, for a web site, given as input to the set-type filtration algorithm the site schema as a tree (ST), a set of web pages from the site (Σ), and the pattern tree (constructed by FiVaTech), the algorithm traverses the schema tree ST in a post-order traversing fashion (lines 1-2). For each set-type node $c$ (line 4), we use the instances of $c$ from the collection of pages Σ to handle it by calling (in order) the two methods "*handleIncorrectType*" and "*handleIncompleteType*".

The first method *handleIncorrectType* shown in Figure 8, if $c$ is an incorrect set type, divides the instances of the type $c$ into different clusters (lines 4-13) such that all instances in each cluster are similar (peer instances) while two instances of different clusters are not similar. The method works as follows. Let $D = Instance(c) = \{x_1, x_2, \dots, x_k\}$ is a set of all instances of $c$, the method starts by having only one cluster $L$=$clust_1$ which has one instance, say $x_1$. It then calculates both $Centroid(clust_i)$ and $Statistical\_Classifier(clust_i)$ for each existing cluster. For each instance $x_i$ ($i$=2, ..$k$), the method either updates the existing clusters by adding $x_i$ to one of them (line 7) or creates a new cluster $clust_i$ which contains the new instance $x_i$ (line 10). Again, $Centroid(clust_i)$ and $Statistical\_Classifier(clust_i)$ for each cluster $clust_i$ is re-calculated after the cluster is updated/created. The instance $x_i$ is added to the cluster $clust_i$ based on the similarity values $Centroid\_Sim$ and $Statistical\_Sim$ between the instance $x_i$ and $clust_i$. After all instances are processed, the method defines a new type $\tau_i$ for each cluster $clust_i$, and then the subtree from the node $c$ in the schema tree ST is replaced by different sibling subtrees of roots $\tau_1, \dots, \tau_k$ (line 16). Finally, we modify the types of the instances in the DOM trees and replace them with the new schema types (line 17).

The method *handleIncompleteType* merges all peer schema types on the left hand side of the current node $c$ (the nodes $\tau_1, \dots, \tau_k$ that replace $c$). Let $c_1$ and $c_2$ are two node types, the method merges them if they are peer types (instances of the second type are similar to the instances of the first one). At this time, the method identifies $c_1$ as an incomplete type, deletes the type $c_2$ from ST, and finally identifies all the instances of $c_2$ in Σ as instances of $c_1$ (i.e., $Instance(c_1)$= $Instance(c_1) \cup Instance(c_2)$). Also, each node in the subtree from $c_2$ is merged to its corresponding peer node in the subtree from $c_1$.

To clarify the proposed filtration algorithm, we apply the algorithm on the schema example shown in Figure 6. As shown in Figure 7, the algorithm handles the two nodes $\{\}_2$ and $\{\}_6$ in the schema tree in Figure 6(b). For the node $c$=$\{\}_2$, the method *handleIncorrectType* divides the instances $\{i_1, i_2, i_3\}$ of the node $c$ into two different clusters $clust_1$=$\{i_1, i_2\}$ and $clust_2$=$\{i_3\}$. As shown in Figure 6, the two instances $i_1$ and $i_2$ in $clust_1$ are similar, while $i_3$ is not similar to both $i_1$ and $i_2$. So, the set type node in the schema tree is replaced by the two types $\tau_1$ and $\tau_2$ that correspond to the two clusters $clust_1$ and $clust_2$, respectively. The type $\tau_1$ is identified as a set type because it has more than one instance in a web page, while $\tau_2$ is identified as a tuple because it has one instance ($i_3$) in each page. The method *handleIncompleteType* does not make any merging among the different types because there is no sibling node on the left hand side. The result of the algorithm after processing the set type node $\{\}_2$ in the schema tree in Figure 6(b) is shown in Figure 9(a). Similarly, the set-type node $\{\}_6$ in the schema tree in Figure 6(b) is divided into the two set-types $\tau_3$ and $\tau_4$ that correspond to the two clusters $clust_3$ = $\{i_5, \dots, i_{12}\}$ and $clust_4$ = $\{i_{13}, i_{14}\}$, respectively, after calling the method *handleIncorrectType* (see Figure 9(b)). Moreover, the method *handleIncompleteType* conquers the two set types $\tau_1$ and $\tau_4$ in Figure 9(b) by removing the type $\tau_4$ from the schema tree and identifying $Instance(\tau_1) = \{i_1, i_2\} \cup \{i_{13}, i_{14}\}$. Also, it removes $\tau_2$ and identifies $Instance(\tau_3) = \{i_5, \dots, i_{12}\} \cup \{i_3\}$. The schema tree in Figure 9(c) is the result of applying the algorithm on the schema tree in Figure 6(b), where $Instance(\tau_1) = \{i_1, i_2, i_{13}, i_{14}\}$ and $Instance(\tau_3) = \{i_3, i_5, \dots, i_{14}\}$.

```
Algorithm handleIncorrectType (c, ST, Σ, patTree)
1.  $D = \{x_1, x_2, \dots, x_k\}$ is set of instances of node c.
2.  $L = \{x_1\}$     // Define one cluster L that includes one instance $x_1$.
3.  $S = \{L\}$     // Define the set of clusters that includes L.
4.  for each instance x in D
5.     for each cluster clust in S
6.        if both $Centroid\_Sim(clust, x) > \tau_1$ and
              $Statistical\_Sim(clust, x) > \tau_2$
              // $\tau_1$ and $\tau_2$ are two thresholds
7.           clust = clust U {x}
8.        else
9.           L={x}
10.          S = S U L
11.       endif
12.    endfor
13. endfor
14. for each cluster clust in S
15.    $\tau$ = identifySchemaType(clust)
16.    addTypeToSchemaTree($\tau$, ST)
17.    changeDOMInstancesofType($\tau$)
18. endfor
```

**Fig. 8: HandleIncorrecteType algorithm.**

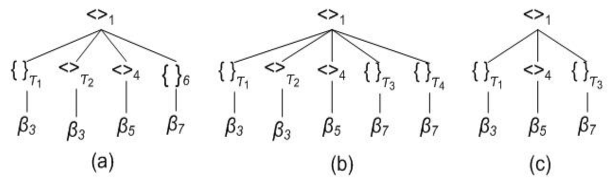
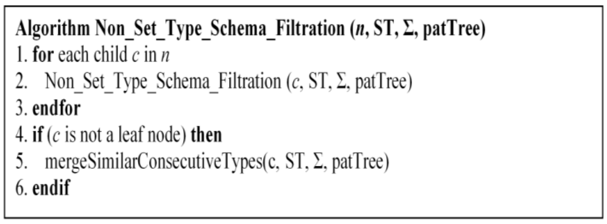


**Fig. 9: The process of filtering out the schema in Fig. 6(b).**

For Step II, combining consecutive non-set similar types, we traverse the schema tree ST in a post-order traversing fashion as shown in Figure 10. For each non-leaf node $c$ (line 4), using the method "mergeSimilarConsecutiveTypes" in line 5, we identify every similar consecutive non-set types and replace them by a new set type $\tau$, and then change the types of the instances of these consecutive types to the new type $\tau$. As a simple example shown in Figure 11(a), if the parent non-leaf type $c$ in the schema tree has 4 child types $\tau_1$, $\tau_2$, $\tau_3$ and $\tau_4$, where the two consecutive types $\tau_2$ and $\tau_3$ are similar (with similar instances), then the algorithm merges the two types $\tau_2$ and $\tau_3$ into a set type $\tau$ as in Figure 11(b). Then, the types of the instances of $\tau_2$ or $\tau_3$ are replaced by the new type $\tau$.



```
Algorithm Non_Set_Type_Schema_Filtration (n, ST, Σ, patTree)
1. for each child c in n
2.    Non_Set_Type_Schema_Filtration (c, ST, Σ, patTree)
3. endfor
4. if (c is not a leaf node) then
5.    mergeSimilarConsecutiveTypes(c, ST, Σ, patTree)
6. endif
```

**Fig. 10: Non-set type filtration algorithm.**

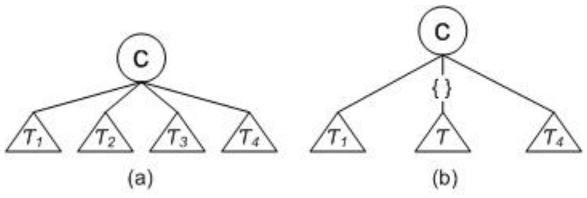


**Fig. 11: An example of similar consecutive non-set schema types.**

## 5. EXPERIMENTS

We conduct three experiments to measure the performance of the proposed classifier and the performance for the two steps of the filtration algorithm. We use a data set of 40 web sites. For each web site, given some pages as input, we run FiVaTech to get as output: the schema of the web site (ST) and the instances of every schema type from the pages.

### 5.1 Classifier Performance for Different Number of Training Pages

In this experiment, we show the performance of the suggested classifier when various number of training pages for each site are used. For each Web site, we fix 5 pages for testing and measure the accuracy of the classifier trained from n (2-5) pages. As shown in Figure 9, the classifier performs better when 3-5 pages have used than when 2 pages are used. Also, a slight increment in the performance is found when 3 to 5 pages have been used.

### 5.2 Set-Type Filtration Performance

In this experiment, we evaluate the performance of the set-type filtration step. For each web site, we manually identify incorrect/incomplete set-type schema types. We use the instances of each set-type to identify whether the set-type is incorrect /incomplete or not. As we discussed in section 4, a set-type is identified as incorrect if there is some instance of this type that is not similar to the type, while the set-type is identified as incomplete if there is some instance which is similar to this type but identified by FiVaTech as an instance of another type. For each schema type τ, we collect all of the instances of the type identified by FiVaTech ($\text{Instances}_{\text{FiVaTech}}$ (τ)) and the instances of the type identified manually ($\text{Instances}_{\text{Manual}}$ (τ)). If the two sets $\text{Instances}_{\text{FiVaTech}}$ (τ) and $\text{Instances}_{\text{Manual}}$ (τ) are equal, we identify the type τ as correct. If $|\text{Instances}_{\text{FiVaTech}}(\tau)|<|\text{Instances}_{\text{Manual}}(\tau)|$, we identify the type τ as incomplete, otherwise, we identify τ as incorrect. Finally, we evaluate the performance of the set-type filtration algorithm by calculating the recall and precision for the schema tree of each web site as follows.

Precision: Precision is the proportion of schema types predicted by the algorithm as incorrect/incomplete that are targets (correctly identified).

Recall: Recall is the proportion of incorrect/incomplete types that are predicted by the algorithm.

Let A is the set of incomplete/incorrect set-types that are identified by the algorithm and B is the set of incomplete/incorrect set-types that are identified manually, so we can define recall and precision as follows:

$$\text{Recall} = \frac{|A \cap B|}{|B|} \quad , \quad \text{Precision} = \frac{|A \cap B|}{|A|}$$

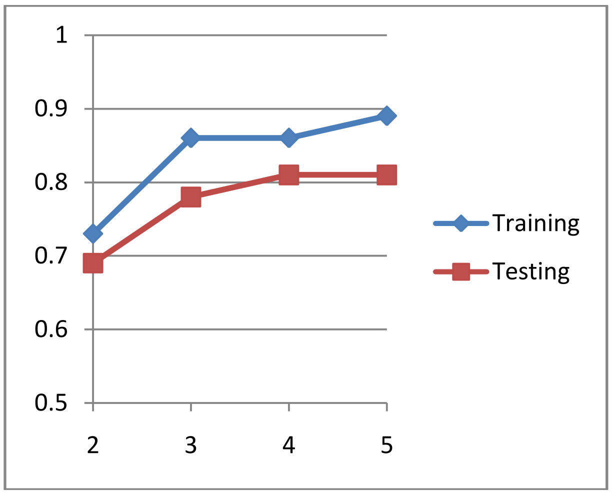


**Fig. 12: The Classifier accuracy for different training pages (2-5) of a Web site.**

The performance of the algorithm with the 40 web sites is shown in Table 3. The experiment, as shown in the table shows an encourage result (recall = 0.85 and precision = 0.56) for the test web sites. In other words, the proposed algorithm can detect/handle reasonable percentage of incomplete/incorrect schema types in the schemas of the test web sites.

### 5.3 Non-Set Consecutive Types Filtration Performance

In this experiment, we evaluate the performance of the second step (consecutive similar non-set types). For each one of the 40 web sites, we count the number of new set-types that should replace consecutive non-set types identified manually and by the algorithm. As we discussed before, every consecutive similar non-set types are replaced by a new set type. Table 4 shows the results for all of the 40 web sites. The performance of the non-set types filtration algorithm is also measured using recall and precision as shown in Table 5, where recall and precision are measured as follows. $\text{Recall} = \frac{|A\cap B|}{|B|}$ , $\text{Precision} = \frac{|A\cap B|}{|A|}$,
where A is the number of new set types identified by the algorithm to replace non-set consecutive types and B is the number of new set-types identified manually. To avoid the cases where either |A| or |B| equal zero, we consider recall = precision = 1.0 if manually there are no consecutive non-set types (|A| = 0) and also the algorithm does not detect any consecutive non-set types (|B| = 0). If one of the two values |A| and |B| equals zero while the other is not equal to zero, we set both recall and precision as zero (recall = precision = 0.0). Also, the experiment as shown in the table gives a reasonable result (recall = 0.58 and precision = 0.55) for the test web sites.

**Table 3: The performance of the set-type filtration algorithm.**

| Web site | Recall | Precision |
|---|---|---|
| YAHP | 0.50 | 0.25 |
| CDPL | 1.00 | 0.50 |
| BUSINESS | 1.00 | 0.67 |
| TELEVISION | 0.78 | 0.64 |
| FOR5 | 1.00 | 0.50 |
| THRV | 0.50 | 0.50 |
| USPA | 0.50 | 1.00 |
| USTX | 1.00 | 0.50 |
| VGAR | 1.00 | 0.50 |
| CESP | 0.67 | 1.00 |
| ALTA | 1.00 | 0.33 |
| EXPE | 0.50 | 0.33 |
| LYCO | 1.00 | 0.20 |
| META | 0.80 | 0.67 |
| NEWS | 1.00 | 1.00 |
| WEBC | 1.00 | 0.50 |
| QUESTION | 0.50 | 0.50 |
| NUMERICAL | 0.80 | 0.67 |
| COMPSKILL | 0.67 | 0.50 |
| JOBS | 0.50 | 0.33 |
| ART | 1.00 | 0.67 |
| COMPONLINE | 0.80 | 0.80 |
| COMPHARDWARE | 1.00 | 0.67 |
| NOKIA | 1.00 | 0.75 |
| BIOLOGICAL | 1.00 | 0.33 |
| OSHISTORY | 1.00 | 0.33 |
| PHOTOS | 0.67 | 0.50 |
| ENGENIRING | 1.00 | 0.25 |
| ANIMAL | 1.00 | 0.67 |
| ENERGY | 0.67 | 1.00 |
| FINANCE | 1.00 | 0.40 |
| HUMAN | 1.00 | 0.50 |
| INDUSTRY | 0.75 | 0.50 |
| INTERNET | 1.00 | 0.25 |
| MATHEMATICAL | 1.00 | 0.67 |
| MUSIC | 1.00 | 0.50 |
| PHYSIC | 0.80 | 0.67 |
| SCIENCE | 0.67 | 0.86 |
| SHOP | 1.00 | 0.50 |
| ECONOMY | 0.75 | 0.50 |
| **Average** | **0.85** | **0.56** |

## 6. CONCLUSIONS

In this paper, we proposed a new versatile classifier for schema type. The classifier can be used as an extractor for web data extraction systems, used for schema mapping and used for post process of the schema generated by web data extraction systems (schema filtration). Since the types in the detected schema have missing names, we use the instances of such types for the classifier construction. Moreover, we used HTML tags contents, DOM trees structural information, and visual information of the schema type instances for the classifier construction. We also defined the schema filtration problem and suggested an algorithm to filter out the schema generated by web data extraction systems. In the future, we aim to use other characterizations of the instance-based schema matching such as relative frequencies of words and combinations of words, phone numbers, zip codes, geographical names, NLP techniques, etc. Also, we plan to extend this work to match elements in different levels of a schema tree or even elements in different schema trees.

**Table 4: Number of new set-types identified manually and by the algorithm.**

| Web site | # of new set-types | |
|---|---|---|
| | **Manual** | **Algorithm** |
| YAHP | 0 | 0 |
| CDPL | 2 | 1 |
| BUSINESS | 1 | 0 |
| TELEVISION | 4 | 3 |
| FOR5 | 0 | 1 |
| THRV | 1 | 2 |
| USPA | 2 | 1 |
| USTX | 0 | 0 |
| VGAR | 1 | 2 |
| CESP | 2 | 3 |
| ALTA | 2 | 1 |
| EXPE | 0 | 1 |
| LYCO | 2 | 1 |
| META | 3 | 4 |
| NEWS | 1 | 2 |
| WEBC | 1 | 0 |
| QUESTION | 2 | 3 |
| NUMERICAL | 3 | 4 |
| COMPSKILL | 5 | 2 |
| JOBS | 1 | 0 |
| ART | 2 | 1 |
| COMPONLINE | 0 | 0 |

| | | |
|---|---|---|
| COMPHARDWARE | 0 | 1 |
| NOKIA | 2 | 3 |
| BIOLOGICAL | 2 | 4 |
| OSHISTORY | 1 | 2 |
| PHOTOS | 1 | 0 |
| ENGENIRING | 0 | 0 |
| ANIMAL | 1 | 1 |
| ENERGY | 2 | 3 |
| FINANCE | 3 | 1 |
| HUMAN | 1 | 2 |
| INDUSTRY | 2 | 0 |
| INTERNET | 1 | 2 |
| MATHEMATICAL | 1 | 0 |
| MUSIC | 2 | 4 |
| PHYSIC | 2 | 1 |
| SCIENCE | 3 | 2 |
| SHOP | 0 | 0 |
| ECONOMY | 3 | 4 |

**Table 5: The performance of the consecutive non-set filtration algorithm.**

| Web site | Recall | Precision |
|---|---|---|
| YAHP | 1.00 | 1.00 |
| CDPL | 0.50 | 1.00 |
| BUSINESS | 0.00 | 0.00 |
| TELEVISION | 0.50 | 0.67 |
| FOR5 | 0.00 | 0.00 |
| THRV | 1.00 | 0.50 |
| USPA | 0.50 | 1.00 |
| USTX | 1.00 | 1.00 |
| VGAR | 1.00 | 0.50 |
| CESP | 1.00 | 0.67 |
| ALTA | 0.50 | 1.00 |
| EXPE | 0.00 | 0.00 |
| LYCO | 0.50 | 1.00 |
| META | 0.67 | 0.50 |
| NEWS | 1.00 | 0.50 |
| WEBC | 0.00 | 0.00 |
| QUESTION | 0.50 | 0.33 |
| NUMERICAL | 0.67 | 0.50 |
| COMPSKILL | 0.20 | 0.50 |
| JOBS | 0.00 | 0.00 |
| ART | 0.50 | 1.00 |
| COMPONLINE | 1.00 | 1.00 |
| COMPHARDWARE | 0.00 | 0.00 |
| NOKIA | 1.00 | 0.67 |
| BIOLOGICAL | 1.00 | 0.50 |
| OSHISTORY | 1.00 | 0.50 |
| PHOTOS | 0.00 | 0.00 |
| ENGENIRING | 1.00 | 1.00 |
| ANIMAL | 1.00 | 1.00 |
| ENERGY | 0.50 | 0.33 |
| FINANCE | 0.33 | 1.00 |
| HUMAN | 1.00 | 0.50 |
| INDUSTRY | 0.00 | 0.00 |
| INTERNET | 1.00 | 0.50 |
| MATHEMATICAL | 0.00 | 0.00 |
| MUSIC | 1.00 | 0.50 |
| PHYSIC | 0.50 | 1.00 |
| SCIENCE | 0.33 | 0.50 |
| SHOP | 1.00 | 1.00 |
| ECONOMY | 0.67 | 0.50 |
| **Average** | **0.58** | **0.55** |

## 7. REFERENCES

[1] Arasu A. and Garcia-Molina H.,“ Extracting Structured Data from Web Pages”, Proc. ACM SIGMOD, pp. 337-348, 2003.

[2] Chang C-H., Kayed M., Girgis M. and Shaalan K., “A Survey of Web Information Extraction Systems”, IEEE Transactions on Knowledge and Data Engineering, vol. 18, no. 10, pp. 1411-1428, 2006.

[3] Kayed M. and Chang C.-H., “Page-level web data extraction from template pages”, IEEE Trans. on Know and Data Eng., vol. 22, no. 2, pp. 249–263, 2010.

[4] Crescenzi V., Mecca G. and Merialdo P., “ RoadRunner: towards-automatic data extraction from large Web sites,” Proceedings of the 26t International Conference on very Large Database Systems (VLDB), Rome, Italy, pp. 109-118, 2001.

[5] Wang and Lochovsky F.,“Data Extraction and Label Assignment for Web Databases”, Proc. Int’l Conf. World Wide Web (WWW-12), pp. 187-196, 2003.

[6] Zhai Y. and Liu B., “Web Data Extraction Based on Partial Tree Alignment”, Proc. Int’l Conf. World Wide Web (WWW-14), pp. 76-85, 2005.

[7] Simon K. and Lausen G., “ViPER: Augmenting Automatic Information Extraction with Visual Perceptions”, CIKM 2005, 2005.

[8] Thamviset W. and Wongthanavasu S., “Information extraction for deep web using repetitive subject pattern”. World Wide Web, August 2013.

[9] Derouiche N., Cautis B., Abdessalem T., "Automatic Extraction of Structured Web Data with Domain Knowledge”. 28th Int. Conference on Data Engineering, pp. 726-737, 2012.

[10] Jinglun G., Zhou Y., Barner K., "View: Visual Information Extraction Widget for improving chart images accessibility”, 19th IEEE Int. Conference on Image Processing, pp. 2865-2868, 2012.

[11] Algergawy A., Nayak R. and Saake G., “Element similarity measures in XML schema matching”, Information Sciences, pp. 4975-4998, 2010.

[12] Milo T. and Zohar S., “Using schema matching to simplify heterogeneous data translation”, Proc. 24th Int. Conf. On Very Large Data Bases, pp. 122–133, 1998.

[13] Rahm E. and Bernstein P., “A survey of approaches to automatic schema matching”, VLDB Journal, vol. 10, no. 4, pp. 334-350, 2001.

[14] Lerner S., “A model for compound type changes encountered in schema evolution”, ACM Trans, Database System, vol. 25, no. 1, pp. 83-127, 2000.

[15] Bertino E., Guerrini G. and Mesiti M., “A matching algorithm for measuring the structural similarity between an XML document and a DTD and its applications”, Information Systems, vol. 29, pp. 23–46, 2004.

[16] Palopoli L., Sacca D., Terracina G. and Ursino D., “Uniform techniques for deriving similarities of objects and sub schemas in heterogeneous databases”, IEEE Trans. Knowledge. Data Eng, vol. 15, no. 2, pp. 271-294, 2003.

[17] Yeh P., Porter B. and Barker K., “Using transformations to improve semantic matching”, in: Proceedings of K-CAP’03, Sanibel Island, FL, pp. 180-189, 2003.

[18] Noy N. and Musen M., “The PROMPT suite: interactive tools for ontology merging and mapping”, J. Hum .Computer. Stud, vol. 59(6), pp. 983-1024, 2003.

[19] Fellbaum C., “WordNet: An Electronic Lexical Database”, The MIT Press, Cambridge, MA, 1998.

[20] Lenat D., “CYC: a large-scale investment in knowledge infrastructure”, Commun. ACM, vol. 38, no. 11, pp. 33-38, 1995.

[21] Berlin J. and Motro A., “Database schema matching using machine learning with feature selection”, CAISE 2002, Toronto, ON, pp. 452-466, 2002.

[22] Li W., Clifton C. and Liu S., “Database integration using neural networks: implementation and experiences”, Knowledge. Inf. Syst, vol. 2(1), pp. 73-96, 2000.

[23] Mirbel I., “Semantic integration of conceptual schemas”, Data & Knowledge Engineering, vol. 21, no. 2, pp. 183-195, 1997.

[24] Madhavan J., Bernstein P. and Rahm E., “Generic schema matching with cupid”, 27th Int. Conferences on Very Large Databases, pp. 49–58, 2001.

[25] Benkley S., Fandozzi J., Housman E. and Woodhouse G., “Data element tool-based analysis (DELTA)”, The MITRE Corporation, Bedford, MA, Technical Report, MTR 95B0000147, 1995.

[26] Zhao H. and Ram S., “Clustering schema elements for semantic integration of heterogeneous data sources”, J. of Database Management, vol. 15(4), pp. 88–106, 2004.

[27] Zhao H. and Ram S., “Clustering similar schema elements across heterogeneous databases: a first step in database integration”, in: K. Siau (Ed.), Advanced Topics in Database Research, Idea Group Publishing, vol. 5, pp. 235–256, 2006.

[28] Algergawy A., Schallehn E. and Saake G., “A Schema Matching-based Approach to XML Schema Clustering”, Linz, Austria. ACM 978 1-60558-349-5/08/0011, November pp. 24-26, 2008.

[29] Kushmerick N., "Wrapper Verification," World Wide Web Journal, vol. 3, no. 2, pp. 79-94, 2000.