DOI 10.1007/s10115-004-0188-z
Springer-Verlag London Ltd. © 2005
Knowledge and Information Systems (2005)

Knowledge and Information Systems

Web data extraction based on structural similarity

Zhao Li, Wee Keong Ng, Aixin Sun

Centre for Advanced Information Systems, School of Computer Engineering, Nanyang Technological University, Singapore

Abstract. Web data-extraction systems in use today mainly focus on the generation of extraction rules, i.e., wrapper induction. Thus, they appear ad hoc and are difficult to integrate when a holistic view is taken. Each phase in the data-extraction process is disconnected and does not share a common foundation to make the building of a complete system straightforward. In this paper, we demonstrate a holistic approach to Web data extraction. The principal component of our proposal is the notion of a document schema. Document *schemata* are patterns of structures embedded in documents. Once the document schemata are obtained, the various phases (e.g. training set preparation, wrapper induction and document classification) can be easily integrated. The implication of this is improved efficiency and better control over the extraction procedure. Our experimental results confirmed this. More importantly, because a document can be represented as a *vector* of schema, it can be easily incorporated into existing systems as the fabric for integration.

Keywords: Classification; Clustering; Framework; Web data extraction

1. Introduction

Large quantities of semistructured documents are appearing on the Web. In recent years, Web data-extraction techniques have been applied in many automatic agent systems, such as price comparison and recommendation systems. They access Web documents to extract and integrate data and to provide data services to users. Unlike free-text documents, semistructured documents have embedded structures. However, there is no explicit schema that comes with these documents, making them difficult to be processed automatically.

Much research has been done in generating Web data-extraction systems that query Web documents and transform them into structured data, such as relational

Received 3 August 2004
Revised 15 October 2004
Accepted 24 October 2004
Published online 2 February 2005

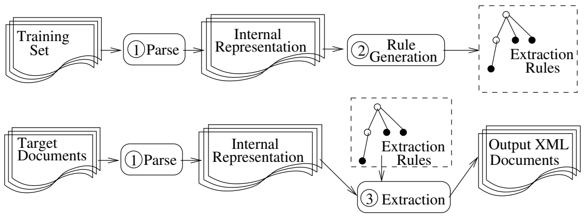


Fig. 1. Web data-extraction process

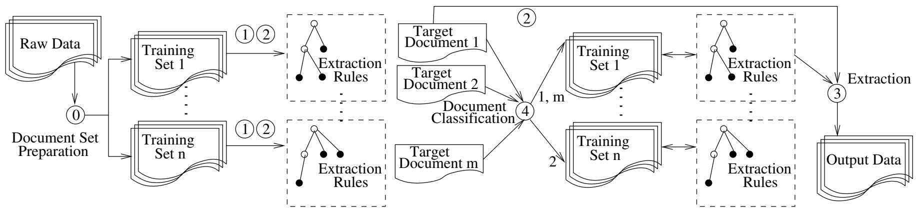


Fig. 2. Enhanced web data-extraction process

data or XML documents with explicit XML schema. Figure 1 presents a typical Web data-extraction process consisting of three phases:

- Phase 1: Documents in training set are parsed to structural representations such as graphs, trees, etc. A training set is a set of documents sharing similar structures.
- Phase 2: Extraction rules are induced from the training set. Extraction rules include knowledge about structure patterns appearing in these documents.
- Phase 3: Given the target set consisting of documents to be extracted, parts in these documents matching with structure patterns are extracted and transformed to structured format. Documents in the target sets are assumed to have similar structures with documents in the training set.

Earlier work in Web data extraction suffers from three problems. First, the training set and the target set are assumed to be predefined. Given a large set of documents that do not share similar structures, multiple training sets have to be constructed and extraction rules are learnt from these training sets independently. To the best of our knowledge, automatical construction of training sets has not been addressed. Second, the learnt extraction rules cannot be applied to the documents that are not from the predefined target set. That is, mapping between a document and the corresponding extraction rules cannot be achieved automatically. Third, most current approaches are ad hoc in terms of the entire Web extraction process. In other words, no consistent view over all phases in Web data extraction is provided. In this paper, we enhance the Web data- extraction process by adding document preparation and document classification processes. A framework is therefore proposed to provide a consistent view over all phases in the enhanced Web data-extraction process. Our contributions are summarized as following:

- The Web data-extraction process is enhanced and formalized as shown in Fig. 2. In this enhanced process, phase 0 automatically clusters structurally similar documents together to construct multiple training sets. Given a new document, phase 4 finds the most structurally similar training set using classification techniques. The extraction rules learnt from the set are then used to extract data from the document.

- We propose a three-layer framework consisting of *instance layer*, *schema layer* and *operation layer*. The instance layer provides a uniform virtual model for Web-data resources, especially HTML and XML documents. The schema layer contains descriptions and attributions of items in the instance layer and provides a common foundation for operations of *training-set preparation*, *document classification* and *data extraction* in the operation layer. These operations handle the instance layer in terms of items in the schema layer.
- To present the feasibility of this framework, an efficient algorithm to build the schema layer and the instance layer is introduced. How to implement training-set preparation, document classification and data extraction in the operation layer is presented. We define notions to represent documents based on the schema layer. Given documents in this representation, it is possible to quantitatively measure the similarity among documents and automatically cluster similar documents together to construct training sets. Data-extraction operations are implemented as extended XQuery operations on the schema layer.

The following sections are organized as follows: Section 2 briefly surveys related work addressing various phases in Web data extraction. Section 3 overviews our Web data-extraction framework and formulates important problems in the framework. In Sect. 4, we define classes of schemata of semistructured documents. Efficient algorithms to detect these schemata are introduced. Section 5 proposes notions to represent semistructured documents using schema information. Based on these notions, we introduce how to measure document similarity and manage documents. Section 6 describes how to extract Web data based on schema information of documents. After presenting experiments and analyzing experimental results in Sect. 7, we conclude our research in Sect. 8.

2. Related work

Much research has been done in Web data extraction. Most work focuses on how to generate extraction rules from a given training set; i.e. the phases 1 and 2 in Fig. 1. This work can be basically classified into three approaches, namely, *manual rule construction*, *annotated document learning* and *unannotated document learning*.

Manual rule construction includes VDB (Virtual Database) (Gupta et al. 1998), Lixto (Baumgartner et al. 2001), Wiccap (Liu et al. 2002) etc. VDB manually generates extraction rules from training documents; thus, rule generation is time consuming. Lixto and Wiccap focus on providing visual interfaces to aid extraction rule generation. Extraction rules generated from these methods are not scalable.

Annotated document learning infers common structures from annotated documents and generates extraction rules automatically. Examples are WIEN (Wrapper Induction for Information Extraction) (Kushmerick et al. 2000, 2002), T-Wrapper (Sakamoto et al. 2001) and TreeAutomata (Kosala et al. 2003). WIEN automatically induces relatively simple structure patterns from training sets containing tabular contents. T-Wrapper extends WIEN and records path information to locate tabular data in Web documents. Nevertheless, these methods are not capable of dealing with documents with complex structures. The TreeAutomata method introduced by Kosala exploits tree automata to extract data from documents. It studies a single annotated Web document and uses a g-testable algorithm to induce automata grammar that recognizes target documents. As the grammar learnt is from a single document, it is too sensitive to noise data.

To avoid user effort in annotating Web documents, unannotated document learning focuses on learning extraction rules from unannotated documents. Work using this approach includes IEPAD (Information Extraction based on Pattern Discovery) (Chang et al. 2001), RoadRunner (Crescenzi et al. 2001), Skeleton (Rajaraman et al. 2001) and ExAlg (Arasu et al. 2003). IEPAD exploits the PATRICIA (Practical Algorithm to Retrieve Information Coded in Alphanumeric) tree algorithm to construct a suffix tree and detect frequent subtrees. RoadRunner devises a method to detect the most common regular expression over HTML strings. Skeleton and ExAlg both devise their own heuristic methods to guess which parts of Web documents are sensitive to users' requirements. Their methods can be divided into three steps: (1) tokenize documents in the training set, (2) count the occurrence of each token in the training set and (3) reconstruct those tokens that appear to fulfill certain requirements into skeletons (template of documents). All the above methods need users to manually prepare training sets.

Compared with work addressing the phases of parsing and rule generation, less work addresses the extraction phase. Kosala et al. (2003) introduce how to apply TreeAutomata to interpret extraction rules. Gottlob et al. (2000) formally analyze expressive capability of extraction rules and optimize performance of extraction in Fig. 1. However, it is hard to apply these methods to the extraction rules learnt by the methods that focus on parsing and rule generation phases because of the lack of a comprehensive framework providing a consistent view over all those problems.

To solve problems in the phases of document-set preparation and document classification, it is reasonable to cluster and classify documents based on structural similarity. Some recent work (Flesca et al. 2002; Nierman et al. 2002) addresses the problem of structural similarity measurement among semistructured documents. Zaki et al. (2003) and Wang et al. (2004) introduced some initial work on document clustering and classification based on structure. However, the structure patterns used in these structural methods do not fit the requirements of Web data extraction; i.e. a pattern should match structures that contain similar semantic information to be extracted.

3. Web data-extraction framework

In this section, we suggest a comprehensive framework that provides a consistent view over various phases in Web data extraction. The problems in building components in this framework are formally defined.

3.1. Overview of web data-extraction framework

Our framework draws ideas from relational databases. A relational database contains at least a relational data model and relational operations. A relational data model includes *data instances (two-dimensional tables consisting of sets of tuples) and* schemata (descriptions of those instances). Relational operations are based on sets; i.e. the inputs and returned results of them are tables instead of individual tuples. Tuples in a relational database may have different structures and can be changed by operations. However, the schema of a table is static. Thus, it is possible to provide a consistent way for all those operations to handle tuples in the system.

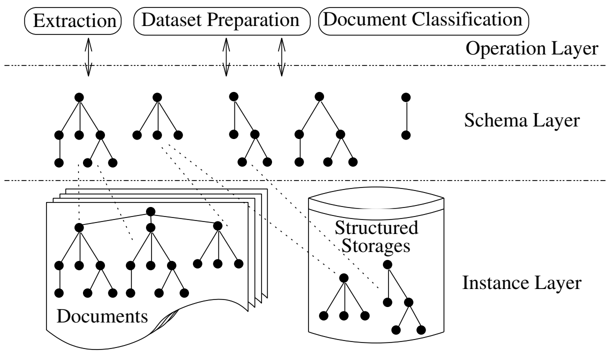


Fig. 3. Our framework for web data extraction

We adapt the relational framework in our Web data-extraction framework (Fig. 3). As our framework works on semistructured data, it has substantial differences from the relational framework. Relational schemata only describe linear tuples, while in our framework, schemata should describe structures of various trees. A relational instance is a table with a unique schema. In our framework, a data instance may be a document, a fragment of a document or a piece of extracted structured data. A document may correspond to multiple schemata, and a schema may correspond to multiple data instances. In Fig. 3, these dashed lines among data instances and schemata represent corresponding relationships among them.

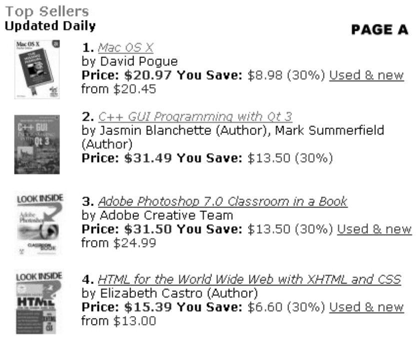
The core of our framework is the *schema layer*. Schemata describe data instances and provide information to the operation layer. To put our framework into practical systems, there are some interesting problems to address; e.g. how to detect these schemata. In the following part of this section, we formally define concepts in the framework and formulate the problem of building each layer.

3.2. Instance layer

In this section, we propose a model to represent Web documents. Based on this representation, the problem of building instance layers is defined.

Semistructured documents are usually modeled as directed graphs (Wang et al. 2004). For example, an HTML document can be parsed into a directed graph; each tag element is parsed to a node and corresponding to each pair of parent–child tag elements, there is a directed edge. Each hyperlink, which describes non-parent–child link relationship between two tag elements, can also be parsed into a directed edge. However, as hyperlinks carry less information in Web data extraction, we do not consider them during the process of parsing in this paper. Without considering hyperlinks, documents can be parsed to trees. Here are some examples:

Example 3.1. Figure 4(a) shows a Web page fragment from the Amazon website on 30 January 2004. It lists four top sellers of computer books. The topmost seller in this page is rendered from the HTML codes as shown in Fig. 5(a). DOM (Document Object Model) trees generated from the codes are shown in Fig. 5(b) (for simplicity, we have not drawn all nodes in the DOM tree; nodes with *folder* icon are nontext nodes and nodes with *paper* icon are text nodes). The right-hand side of pages B and C in Fig. 4(b) are detailed information about the two topmost hot books, respectively.



Top Sellers
Updated Daily
PAGE A

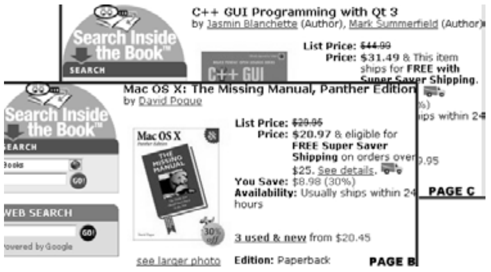
1. *Mac OS X*
by David Pogue
Price: $20.97 You Save: $8.98 (30%) Used & new
from $20.45

2. *C++ GUI Programming with Qt 3*
by Jasmin Blanchette (Author), Mark Summerfield (Author)
Price: $31.49 You Save: $13.50 (30%)

3. *Adobe Photoshop 7.0 Classroom in a Book*
by Adobe Creative Team
Price: $31.50 You Save: $13.50 (30%) Used & new
from $24.99

4. *HTML for the World Wide Web with XHTML and CSS*
by Elizabeth Castro (Author)
Price: $15.39 You Save: $6.60 (30%) Used & new
from $13.00

(a) Top seller books



C++ GUI Programming with Qt 3
by Jasmin Blanchette (Author), Mark Summerfield (Author)
List Price: ~~$44.99~~
Price: $31.49 & This item ships for **FREE with Super Saver Shipping.**

Mac OS X: The Missing Manual, Panther Edition
by David Pogue
List Price: ~~$29.95~~
Price: $20.97 & eligible for **FREE Super Saver Shipping** on orders over $25. See details.
You Save: $8.98 (30%)
Availability: Usually ships within 24 hours

3 used & new from $20.45

see larger photo **Edition:** Paperback **PAGE B**

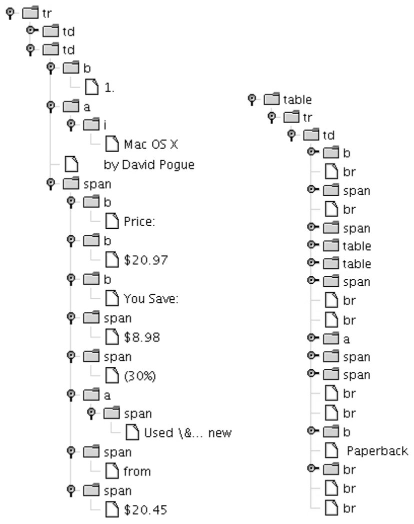
PAGE C

(b) Book details

Fig. 4. Sample pages

```
<tr>
 <td align="right">
  <a href="/exec/...">
  <img src="http://ima..."/></a></td>
 <td class="small">
  <b>1.</b>
  <a href="/exec/...">
   <i>Mac OS X</i></a>
  by David Pogue
  <span class="small">
   <b>Price:</b>
   <b class="price">$20.97</b>
   <b>You Save:</b>
   <span class="price">$8.98</span>
   <span class="price">(30%)</span>
   <a href="http://ww...">
   <span class="small">Used &amp;... new</span>
   </a>
   <span class="tiny">from</span>
   <span class="price">$20.45</span>
  </span>
 </td>
</tr>
```

(a)



(b) (c)

Fig. 5. An HTML document fragment and DOM trees

Figure 5(c) shows the top four levels of the DOM tree corresponding to the HTML codes for the right-hand side of page B.

In Example 3.1, an element name is mainly used to describe page layout. However, for some semistructured documents, especially XML, an element name may contain important information. Thus, in this paper, we model semistructured documents as labeled trees—document tree—as defined below:

Definition 3.1 (Document tree). A document tree is a rooted, labeled tree that can be defined as a four-tuple: $t = \langle V, E, r, \gamma \rangle$, where V is the set of nodes corresponding to tag elements, E is the set of edges connecting these nodes, r is the root node, $\gamma : V \rightarrow L$ is a function that assigns each node a string label, where L is the label set of t. An edge e is an ordered two-tuple (u, v), where $u, v \subseteq V$ and u is the

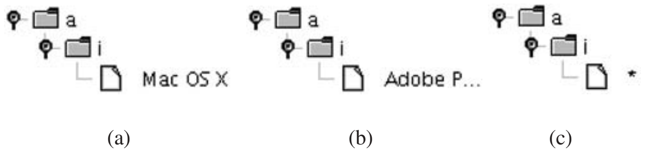


Fig. 6. Instances and their schema

parent node of node v. Root node r has no parent node. Each nonroot node u in V has exactly one parent node.

In this paper, we restrict our discussion to HTML and XML documents so that we can exploit DOM parsers to parse documents to trees. During the parsing processing, we label each nontext node with the name of the corresponding element in the original documents and label each text node with its value.

The objective of Web data extraction is to extract fragments from document trees that are relevant to a user's requirements. We refer to these fragments as data instances, as defined below:

Definition 3.2 (Data instance). Given a document tree $t = \langle V, E, r, \gamma \rangle$, $t_1 = \langle V_1, E_1, r_1, \gamma_1 \rangle$ is a data instance (DI) of t if $V_1 \subseteq V$, $E_1 \subseteq E$ and $\gamma_1 = \gamma$. This relation is denoted as $t_1 \subseteq t$. Given two Dis, t_i and t_j, t_i is a sub-DI of t_j if $t_i \subseteq t_j$.

Example 3.2. In Example 3.1, the document tree in Fig. 5(b) is a DI of the document tree corresponding to page A in Fig. 4(a). This instance corresponds to the first book entry in page A. Figure 6(a) is a DI of Fig. 5(b). Figure 6(a) corresponds to the title of the first book, while Fig. 6(b) corresponds to the title of the third book.

Definition 3.2 states that a document tree is also a DI. We therefore do not distinguish between semistructured document, document tree and data instance in the remaining sections of this paper. Given the definition of *DI*, the problem of *building instance layer* is quite straightforward; i.e. given n Web documents $\mathcal{P} = \{p_1, \ldots, p_n\}$, induce the set of DIs $\mathcal{D} = \{d_1, \ldots, d_n, d_{n+1}, \ldots, d_{n+m}\}$, where d_i is parsed from p_i and $d_{n+j} \in \mathcal{D}$ if and only if $d_{n+j} \subseteq d_k$, $k \in [1, n]$.

3.3. Schema layer

In this section, we formally define schema and discuss problems in building the schema layer.

3.3.1. Schema

Repeated contents in semistructured documents are usually prominent and easily raise a reader's attention. Most Web data-extraction systems (Arasu et al. 2003; Chang et al. 2001; Crescenzi et al. 2001) assume that repeated contents are important and should be extracted. For example, in the example in Fig. 4, all books have similar formats and most parts of their HTML codes are repeated, like those in Fig. 5. These repeated contents can be parsed to equivalent DIs, as defined below:

Definition 3.3 (Equivalence). Given two Dis, $t_1 = \langle V_1, E_1, r_1, \gamma_1 \rangle$ and $t_2 = \langle V_2, E_2, r_2, \gamma_2 \rangle$, t_1 is *equivalent to* t_2 if and only if there exists a bijection $\mathcal{M}$ between V_1 and V_2 such that

- $\mathcal{M}(r_1) = r_2$,
- $(u, v) \in E_1$ if and only if $(\mathcal{M}(u), \mathcal{M}(v)) \in E_2$,
- $\gamma_1(u) = \gamma_2(\mathcal{M}(u))$.

This definition does not consider the orders among sibling nodes; i.e. a document tree is an unordered tree. In Sect. 4, we shall introduce the benefit of modelling HTML documents and other semistructured documents as unordered trees.

The two instances in Figs. 6(a) and (b) are almost the same, except for the labels of two text nodes. If we use a label "*" (wildcard character) to replace both labels, the two instances will become Fig. 6(c). A wildcard character "*" is a regular expression that generates any string, denoted as $* \vdash \ell$, where ℓ is a label. We assume "*" does not appear in label sets of DIs of document trees to be extracted. As introduced before, schemata are objects describing a set of DIs. Thus, we treat the instance in Fig. 6(c) as a schema. We define a schema below:

Definition 3.4 (Schema). A schema s is a five-tuple $\langle V, E, r, \gamma, A\rangle$, where $\langle V, E, r, \gamma\rangle$ is a DI and γ labels some nodes with "*". A is an n-tuple $\langle a_1, a_2, ..., a_n\rangle$, where a_i is an attribute, $i \in [1, n]$.

Attributes in a schema are important to aid data extraction and will be discussed in Sect. 3.3.3. Here, we may assume $A = \langle |s| \rangle$, where $|s|$ denotes the size of a schema s; i.e. the cardinality of the node set V of s.

Definition 3.5 (Conformation and type). A DI $t = \langle V, E, r, \gamma\rangle$ conforms to a schema $s = \langle V_s, E_s, r_s, \gamma_s, A_s\rangle$ or s is the schema of t, if and only if there exists a bijection $\mathcal{M}$ between V and V_s such that

- $\mathcal{M}(r) = r_s$,
- $(u, v) \in E$ if and only if $(\mathcal{M}(u), \mathcal{M}(v)) \in E_s$,
- $\gamma(u) = \gamma_s(\mathcal{M}(u))$ or $\gamma_s(\mathcal{M}(u)) \vdash \gamma(u)$,

where t conforms to s, denoted as $t \succcurlyeq s$. A set of DIs conforming to the same schema is known as a *type*.

3.3.2. Schema detection

In our framework, the instance layer contains all Web documents to be extracted. The schema layer contains very useful information of DIs in the instance layer. For example, two documents are structurally similar if the sets of schemata corresponding to them are the same. Given a set of n document trees $\{d_1, \ldots, d_n\}$ that contain a set of Dis, $\mathcal{D} = \{t_1, \ldots, t_m\}$, *schema detection* is the procedure of inducing the set of schemata $\mathcal{S} = \{s_1, \ldots, s_n\}$, where $s_i \in \mathcal{S}$ if and only if $d_j \in \mathcal{D}$ and $d_j \succcurlyeq s_i$.

The problem of schema detection is to find the set of schemata of all DIs in documents (or subtrees of document trees). This problem can be reduced from the problem of *the largest common subtrees (LCST)* (Akutsu et al. 1992). As LCST is P for two trees and NP-hard for more than two trees (Akutsu et al. 1992), the schema detection problem is also NP-hard. The same is applied to the problem of building instance layers. Some approximate solutions of similar problems have been proposed by putting some restrictions on subtrees to be processed; e.g. TreeMiner (Zaki et al. 2003) only processes subtrees of which the frequency of occurrence exceeds a threshold. We define a constrained version of this problem in Sect. 4 and an efficient algorithm is proposed to solve it.

3.3.3. Attributes in schema

Some existing Web data-extraction systems (Arasu et al. 2003; Chang et al. 2001) assume large structure patterns that match large numbers of structures in documents are important; contents that match these patterns are then extracted. Similarly, we compute schema weight to measure the importance of corresponding DIs based on two observations.

Observation 3.1. A type including a large number of DIs is usually important in documents, and its corresponding schema is important.

Observation 3.2. A schema with large size is usually important in documents.

There are two factors influencing the weight of a schema—the cardinality of its corresponding type and its size. Given a document tree t, T is the type in t and s is a schema conformed by DIs in T, the *weight* of s in the document is:

$$\omega(s) = \ln \|T\| \times |s|, \tag{1}$$

where $\|T\|$ is the cardinality of T, also known as the *document frequency* of s in this paper.

The weight of a schema is important evidence to decide which DIs should be extracted. However, it is not enough to decide which DIs should be extracted based on schema weight only. One property of semistructured documents is irregularity—DIs conforming to the same schema may encode different kinds of content. Another property of semistructured documents is that there exist redundant contents, e.g. advertisement bars appearing in many HTML pages. In this section, we introduce attributes of schemata that provide additional information other than the schema weight. Users may control extraction operations based on these attributes.

Given a document set $\mathbb{C}$, we collect the following attributes of a schema:

- Size, document frequency (DF) and weight.
- Set frequency (SF): Given a set of documents, $\mathbb{C}$, if s is the schema conformed by DIs in type T, SF of s in $\mathbb{C}$ is $|T|$.

When all documents are organized into clusters, each cluster is a set of similar documents. Given these clusters, we may identify more attributes of a schema.

Inverse set frequency (ISF) of a schema s is denoted with $I(s)$. $I(s) = log\frac{N}{n}$, where N is the number of document sets and n is the number of document sets containing DIs conforming to s.

Suppose a document set $\mathbb{C}$ contains types T_1 to T_n, the weight of the schema s conformed by DIs in type T_i is $\varpi(s) = \|T_i\|/max_{j=1}^{n}\|T_j\|$, the *set weight* (SW) of a schema s is

$$W(s) = \varpi(s) \times ISF(s)\,. \tag{2}$$

To be easily accessed by users, Web documents often include much redundant information. For instance, the same navigation bar may appear in many pages. An entropy measurement of fragments in Web documents was suggested in Lin et al. (2002) to detect redundant information, only fragments with small entropy should be extracted. Inspired by this idea, we calculate entropy of schemata and restrict that only DIs conforming to schemata with entropy smaller than a threshold will be extracted. Given n document sets, *entropy* of a schema s is

$$Ent(s) = -\sum_{i=1}^{n} \varpi(s) log \varpi(s)\,. \tag{3}$$

Usually, a type of DIs appearing closely and regularly is likely to be interesting. Based on this observation, we define two attributes of a schema—mean distance (MD) and standard deviation of distance (SDD) to measure how close and regular those DIs in a type are. To compute MD and SDD, we need to know distance and the order relation among DIs and orders among DIs.

Distance between DIs t_i and t_j in a document tree is the number of nodes in the shortest path between root nodes of t_i and t_j. If t_i and t_j belong to different document trees, distance between them is 1.

Given a set of document trees $\{d_1, \ldots d_n\}$, a node r_0 is added as the parent node of these trees; the tree rooted from r_0 is denoted as T. The *position* of a DI, t, is $\theta(t) = m$ if the root of t is the mth nodes accessed in the preorder traversal of T. An *order relation* between a pair of DIs is denoted with $t_i \preceq t_j$. We say $t_i \preceq t_j$ if $\theta(t_i) \leq \theta(t_j)$.

Given a type T, we sort all DIs in T, such that $t_i \preceq t_{i+1}$. We say that t_i and t_{i+1} are adjacent DIs. *Mean distance (MD)* of the schema s conformed by DIs in T, $\mu(s)$, is the mean value of distance between each pair of adjacent DIs belonging to T.

$$\mu(s) = \frac{\sum_{i=1}^{n-1} d(t_i,\, t_{i+1})}{n-1}. \tag{4}$$

Standard deviation of distance (SDD) of s is defined as

$$\sigma(s) = \sqrt{\frac{\sum_{i=1}^{n-1} (d(t_i,\, t_{i+1}) - \mu(s))^2}{n-1}}\,. \tag{5}$$

We store all attributes detected in a schema s in a tuple $\langle$*Size*, *DF*, *Weight*, *SF*, *ISF*, *SW*, *MD*, *SDD*$\rangle$.

We know an important property of semistructured documents is sectional, i.e. contents in different parts of a document may contain different information. Usually, DIs belonging to the same type appear in the same part in a document. Thus, if most DIs belonging to the same type appear in a part and a DI is far away from this part, this DI may be an outlier and need not to be extracted. We define *distance offset* of a DI t_i as

$$\Delta(t_i) = \frac{d(t_{i-1},\, t_i) + d(t_i,\, t_{i+1})}{2\mu(C)} - 1\,. \tag{6}$$

3.4. Operation layer

Once the instance layer and the schema layer are built, we obtain the mapping relationships among DIs and schemata. Series operations are proposed based on these relationships to aid Web data extraction.

A DI and its sub-DIs conform to a set of schemata, i.e. it is possible to use a set of schemata to describe a document. Section 5 discusses how to manage documents based on the set of schemata conformed by DIs appearing in these documents. As a schema may be conformed by a set of DIs, this DI set can be handled in terms of the schema. For example, an operation $o = e(s)$ can be defined to extract all DIs conforming to s. Based on attributes in a schema, o can be extended to more complicated operations. In Sect. 6, we shall introduce how to extend the standard XML query language, XQuery, to implement Web data-extraction operations in the operation layer. The extension of XQuery shows the possibility for our framework to be adopted by other applications. Before discussing the details of building the operation layer, we shall introduce our approach to build the schema layer and the instance layer in the next section.

4. Constrained schema detection

As defined in Sect. 3.3.2, the problems of building instance layer and schema layer are intractable. To solve the problems, we only consider a special class of DIs in document trees based on an observation:

Observation 4.1. Most interesting contents appear near leaf nodes in many document trees. Text information to be extracted is mainly embedded in text elements, especially for HTML documents.

Based on this observation, we define a class of DIs and schemata as follows:

Definition 4.1 (Bottom data instance and bottom schema). Given a document forest F, a bottom data instance (BDI) is a data instance where all leaf nodes are also leaf nodes in F. A bottom schema (BS) is a schema for a BDI that labels and only labels text nodes with “*”.

The only difference between a DI and a BDI is that the leaf nodes of a BDI are also the leaf nodes of the document tree containing the BDI. In the following parts of this paper, we only consider BDI and BS, i.e. we do not distinguish between BDI and DI, BS and schema. Given this restriction, we shall introduce an algorithm with almost linear time complexity to detect all schemata and DIs from given documents and assign a unique ID to each schema. Before introducing our algorithm, we provide an alternative definition of *conformation* to make our algorithm easier to understand.

Definition 4.2 (Conformation). Given a data instance $t = \langle V, E, r, \gamma \rangle$, its *type vector* is $\varphi(t) = \langle \gamma(r), o_1, \ldots, o_n \rangle$, where o_1 to o_n are sorted schema ID of n child DIs of r. A DI $t = \langle V, E, r, \gamma \rangle$ conforms to a schema $s = \langle V_s, E_s, r_s, \gamma_s, A_s \rangle$ if $\varphi(t) = \varphi(s)$.

Note that Definition 4.2 is equivalent to Definition 3.5. From the definition, it is possible to traverse document trees from the bottom to top and detect all schemata in one traversal. Ideally, we hope that document fragments containing the same kind of information can be parsed to the same type of DIs conforming the same schema. Unfortunately, the real situation is more complicated. For example, in Fig. 4(a), the first and the third books in page A are rendered from document fragments that can be parsed into the same type of DIs. This is ideal; however, the second data instance of book information does not conform to the same schema as them. Unlike other books, the book titled “*C++ GUI Programming with QT3*” has no second-hand price. There are some other properties of semistructured documents that make data instances describing the same kind of information different and cannot be put into the same type.

Algorithm 1 LayerBuilder

```
Require: a document forest $F$.
 1: initiate a dequeue of nodes $\mathcal{D}$;
 2: read the schema table $\mathcal{S}$ from the schema layer, each tuple is $[id, s, T]$;
 3: read the type table $\mathcal{T}$ from the instance layer, each tuple is $[id, T, D]$;
 4: push all the leaf nodes in $F$ to $\mathcal{D}$;
 5: while $\mathcal{D}$ is not empty do
 6:    pop the first node $d$ from $\mathcal{D}$, $t$=$TreeRootedFrom(d)$;
 7:    search $\mathcal{S}$ for $\mathcal{S}[i]$: $\varphi(t) \approx \mathcal{S}[i].T$;
 8:    if found then
 9:       insert $t$ to $\mathcal{T}[i].D$;
10:    else
11:       let $id_d = |\mathcal{S}|$, $s$ is $t$'s schema;
12:       insert $[id_d, s, \varphi(t)]$ to $\mathcal{S}$;
13:       insert $[id_d, \varphi(t), \{t\}]$ to $\mathcal{T}$;
14:    end if
15:    search sibling nodes of $d$ in $\mathcal{D}$;
16:    if failed then
17:       push the parent node of $d$ to $\mathcal{D}$;
18:    end if
19: end while
```

- Missing attributes: In Example 3.1, information of a book may include author, price, etc. Sometimes, some books have a second-hand price and others do not have such information.
- Multivalued attributes: A book may have more than one author.
- Disjunctive delimiters: A document may use different delimiters to mark the same attribute. For example, the titles of hot sales or the special price may appear in bold format.

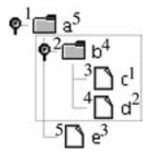
To resolve the problem that the same kind information is embedded in different types, we relax the condition in Definition 4.2 and place two DIs in the same type if most sub-DIs appearing in them are equivalent.

Definition 4.3 (Partial equivalence, conformation and type). Given a DI $t = \langle V, E, r, \gamma \rangle$, there are m sub-DIs t_1 to t_m rooted from m child nodes of root r. The DIs t_1 to t_m conform to schemata s_1 to s_k, respectively. $\{s_1, ..., s_k\}$ is the child schema set of t, denoted as $\mathcal{S}_c(t)$. Given a DI $s = \langle V_s, E_s, r_s, \gamma_s \rangle$, suppose $\mathcal{S}_{in} = \mathcal{S}_c(t) \cap \mathcal{S}_c(s)$ and $\mathcal{S}_{un} = \mathcal{S}_c(t) \cup \mathcal{S}_c(s)$, type vectors $\varphi(V)$ and $\varphi(V_s)$ are partial equivalents if

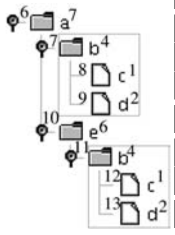
$$\sum_{s_i \in \mathcal{S}_{in}} |s_i| > r \times \left(\sum_{s_i \in \mathcal{S}_{un}} |s_i| \right), \tag{7}$$

denoted as $\varphi(V) \approx \varphi(V_s)$, where $r \in [0, 1]$. DIs t and s are partial equivalents if $\varphi(V) \approx \varphi(V_s)$, denoted as $t \approx s$. If s is a schema, we say t partially conforms to s, denoted as $t \succsim s$. We call a set of DIs that partially conforms to the same schema as a partial type.

Based on the preliminary introduction above, we provide a procedure to build the instance layer and schema layer in Algorithm 1. In the schema layer, the schemata detected in each document set are stored in a schema table. Each tuple in a schema table consists of a schema's ID, the schema and its type vector, denoted as $\langle id, s, T \rangle$. In the instance layer, the partial types detected in each document set are stored in



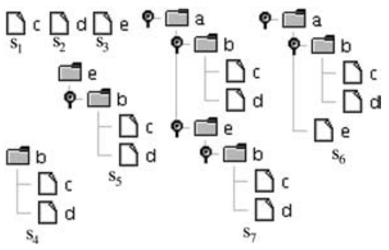
(a) DI_1



(b) DI_2

id	s	T
1	s_1	[c]
2	s_2	[d]
3	s_3	[e]
4	s_4	[b,1,2]
5	s_5	[a,3,4]
6	s_6	[e,4]
7	s_7	[a,4,6]

(c) Final status of $\mathcal{S}$



(d) Schema detected

Fig. 7. Running example

a type table. Corresponding to each partial type, there is a tuple that consists of the schema ID of this type, its type vector and the list of DIs belonging to this type, denoted as $\langle id, T, D\rangle$. Given a document set, Algorithm 1 first initiates the schema table and type table; then reads nodes from bottom to top, while detecting the schema of the DI rooted from each node. New schemata, DIs and types detected will be appended to the schema table and type table correspondingly. As the schema table and type table of a document set are stored persistently, when the document set is changed, e.g. a new document is added, Algorithm 1 only needs to read nodes from changed parts and modify the schema table and type table. We know that DIs in a partial type may partially conform to multiple schemata. Line 13 of Algorithm 1 assigns an *id* to a partial type; this *id* is the same as the *id* of the schema that is conformed by the first DI inserted to the type. We call this schema the *representative schema* of this type.

In Fig. 7, we provide a running example of Algorithm *SchemaDetector*. Here, we set r in (7) to 1. Figures 7(a) and (b) are input Dis, where “a”, “b”, “c”, “d”, “e” are labels of nodes. The number at the left-upper corner of each node is its preorder traversal position in the input document forest. The number at the right-upper corner of each node is the *id* of schema to which the DI rooted from the node conforms. The SchemaDetector performs its operation in the following order:

- Detect schemata of leaf nodes “c”, “d”, “e”, delete corresponding nodes from $\mathcal{D}$, and insert the detected schemata into $\mathcal{S}$, i.e. the first three tuples.
- Detect schema of subtree rooted at “b” as schemata of all its child nodes have been detected, delete node “b” from $\mathcal{D}$, and insert the schema into $\mathcal{S}$, i.e. the fourth tuple.
- Similarly, detect schema of subtrees rooted at “a” (with preorder traversal position 1) and “e”, respectively, delete the two nodes from $\mathcal{D}$ and insert the detected schemata into $\mathcal{S}$, i.e. the 5th and 6th tuples.
- Detect schema of subtree rooted at “a” (with preorder traversal position 10), delete the node from $\mathcal{D}$ and insert the detected schema into $\mathcal{S}$, i.e. the last tuple.

Figure 7(c) is the final status of $\mathcal{S}$.

SchemaDetector accesses each node only once. For a set of DIs containing n nodes, it assigns class *id* to those nodes in n iterations. In the ith iteration, the complexity of all statements except line 7 is $O(1)$. Statement 7 searches in a table, and its complexity is $O(n)$ in the worst case, i.e. all sub-DIs accessed conform to

$$DI_1 = [1\ 1\ 1\ 1\ 0\ 1\ 0]$$
$$DI_2 = [1\ 1\ 1\ 1\ 1\ 0\ 1]$$

(a) Documents

$$M = \begin{bmatrix} 1\ 1\ 1\ 1\ 0\ 1\ 0 \\ 1\ 1\ 1\ 1\ 1\ 0\ 1 \end{bmatrix}$$

(b) Document space

Fig. 8. The vector space of documents

different schema. Thus, the complexity of SchemaDetector is $O(n^2)$ in the worst case. However, in a large document set, the number of all schemata is often much smaller than the number of nodes; thus, the complexity of SchemaDetector is near $O(n)$.

5. Document management

As shown in Fig. 2, the document-set preparation operation clusters structurally similar documents together as training sets. Document classification operation classifies a document to a training set. These two operations significantly improve efficiency of Web data extraction. In this section, we first introduce how to measure structural similarities among documents, followed by the details of these two operations.

5.1. Document similarity measurement

To measure similarity among documents, we first represent them as vectors of schema weights and compute document similarities by computing the similarity among these vectors. Corresponding to each schema embedded in a DI, there is an item in the vector recording the weight of the schema. Suppose the weight of a schema is 1 if the schema appears in a document tree and 0 if it does not appear; the vectors of DI_1 and DI_2 in Fig. 7 are shown in Fig. 8(a). So far, we can represent all documents in a vector space. For example, DI_1 and DI_2 can be represented using the matrix in Fig. 8(b).

Given two documents, t_1 and t_2, suppose there are n representative schemata, we represent $t_1 = \langle \omega_1(s_1), \ldots, \omega_1(s_n) \rangle$ and $t_2 = \langle \omega_2(s_1), \ldots, \omega_2(s_n) \rangle$, where $\omega_i(s_j)$ is the weight of s_j in t_i. The similarity between t_1 and t_2 can be computed using (8).

$$\Omega_C(t_1, t_2) = \frac{\sum_{i=1}^{k}(\omega_1(s_i) \times \omega_2(s_i))}{\sqrt{\sum_{i=1}^{k} \omega_1(s_i)^2 \times \sum_{i=1}^{k} \omega_2(s_i)^2}}. \tag{8}$$

5.2. Document set preparation and document classification

Given the structural similarity among documents, we now introduce how to solve problems in the phases of document-set preparation and document classification.

5.2.1. Document-set preparation

We prepare a training set by clustering similar documents together. In the similarity-calculation phase, we can obtain similarity square matrix $\mathbb{M}_{m\times m}$, where m is the number of document trees and $\mathbb{M}_{i,j}$ is the similarity between document t_i and document t_j. Given the similarity matrix, we choose bisecting k-means clustering algorithm that clusters documents into k clusters (k is predefined by a user), from $\mathbb{C}_1$ to $\mathbb{C}_k$, such that the maximum value of the following formula is obtained:

$$\sum_{i=1}^{k} \sqrt{\sum_{t_1, t_2 \in \mathbb{C}_i} \mathbb{M}(t_1, t_2)} \, . \tag{9}$$

5.2.2. *Document classification*

As introduced before, we may detect schema attributes in terms of document sets and extract data based on those attributes. Given a new document, it is an issue on how to exploit the known attributes to extract data. We propose to classify the new document to an existing training set and apply extraction rules learnt from the set on this document. We referred to XRules (Zaki et al. 2003) for some initial work of documents classification based on tree structures of documents. XRules mines rules, like $T \Rightarrow \mathbb{C}$, that describe the relationship between the appearance of a tree structure t in a document and the document belonging to class $\mathbb{C}$. During the classification phase, XRules combines the evidences of structures appearing in a document and suggests a class to the document. However, XRules does not consider the occurrence frequency of structure t in the documents to be classified. Here, we suggest a more general approach to classify document trees. From the results of the document-set preparation phase, we use a vector, $\mathbb{W}_{\mathbb{C}} = \langle \omega_{\mathbb{C}}(s_1), \ldots, \omega_{\mathbb{C}}(s_n) \rangle$ to represent a training set, where $\omega_{\mathbb{C}}(s_i)$ is the weight of a representative schema s_i in training set $\mathbb{C}$. Previously, we have introduced how to represent a document tree using a weighted vector. Assume that $\mathbb{W}_t = \langle \omega_t(s_1), \ldots, \omega_t(s_n) \rangle$ is the weight vector of document tree t, the similarity between a document tree and a document set is

$$\Omega_S(t, \mathbb{C}) = \frac{\sum_{i=1}^{k} (\omega_t(s_i) \times \omega_{\mathbb{C}}(s_i))}{\sqrt{\sum_{i=1}^{k} \omega_t(s_i)^2 \times \sum_{i=1}^{k} \omega_{\mathbb{C}}(s_i)^2}} \, . \tag{10}$$

We may classify a document tree t to the training set that is most similar if the similarity value is larger than a threshold; otherwise, t should be treated as an outlier.

6. Data extraction

In this section, we discuss the details of how to extract data based on schemata.

In Sect. 3.3.3, attributes in a schema are defined. We may directly output DIs with large weight or fulfilling some other requirements. However, the requirements from users are not unitary. It is more flexible to allow a user to use a predefined threshold on each attribute to control extraction results. In the original documents, we annotate a DI with schema attributes by adding attributes to the root element of the DI. Future queries on annotated documents in terms of schemata are allowed. For example, XML document in Fig. 9(a) is the original document of document trees in Fig. 7. It may be annotated to Fig. 9(b).

Given attributes in schemata, a user may set threshold to instruct the system to extract only those DIs conforming to schemata with special attribute values. We

```
<a>
 <b>
  <c/>
  <d/>
 </b>
 <e>
  <b>
   <c/>
   <d/>
  </b>
 </e>
</a>
```

(a)

```
<a schema="s7" size="8" DF="1" ...>
 <b schema="s4" size="3" DF="2" ...>
  <c schema="s1" size="1" DF="2" .../>
  <d schema="s2" size="1" DF="2" .../>
 </b>
 <e schema="s6" size="4" DF="1" ...>
  <b schema="s4" size="3" DF="2" ...>
   <c schema="s1" size="1" DF="2" .../>
   <d schema="s2" size="1" DF="2" .../>
  </b>
 </e>
</a>
```

(b) "out.xml"

Fig. 9. Annotated document

```
define  function DF($e as element,
     $t as xs:decimal)  as xs:boolean
{
if ($e/DF>$t)
then true()
else false()
}
{-- Function A: Evaluate Frequency --}

define  function Size($e as element,
     $t as xs:decimal)  as xs:boolean
{
if ($e/@Size>$t)
then true()
else false()
}
{-- Function B: Evaluate Size --}

define function Extract($e as element)
as element*
{
if (Size($e, 2) and DF($e, 1)) then
( $e )
else
( for $child in $e/*
    let $b := Extract($child)
return $b )
}
{-- Function C: Extract --}
```

Fig. 10. Extension function for XQuery

exploit the standard XML query language, XQuery, to extract final results from annotated documents. Here, we give an example of extending XQuery to allow users to extract data based on schema information. Function A in Fig. 10 is to judge whether the DF of a DI's schema is larger than the given DF. Function B is to judge whether the size is larger than the given size. Function C recursively accesses nodes from root to leaf and filters all those nodes from root to these DIs conforming to schemata that are larger than 2 and have DF larger than 1.

Given these extension functions, users may exploit very simple XQuery scripts to extract final data from annotated documents. For example, users may want to extract records conforming to large-size schema with high DF. Submitting XQuery

```
<FinalData>
{Extract(document("out.xml")/a}
</FinalData>
{-- Query A --}

<FinalData>
 <b schema="$s_4$" size="3" DF="2" ...>
  <c schema="$s_1$" size="1" DF="2" .../>
  <d schema="$s_2$" size="1" DF="2" .../>
 </b>
 <b schema="$s_4$" size="3" DF="2" ...>
  <c schema="$s_1$" size="1" DF="2" .../>
  <d schema="$s_2$" size="1" DF="2" .../>
 </b>
</FinalData>
{-- Final Results --}
```

Fig. 11. Extended XQuery and results

Table 1. Data sets

Name	URL
IEPAD	http://140.115.156.70/iepad/
RoadRunner	http://www.dia.uniroma3.it/db/roadRunner/index.html
RISE	http://www.isi.edu/info-agents/RISE/
WIEN	http://www.cs.ucd.ie/staff/nick/home/research/wrappers/
ExAlg	http://www-db.stanford.edu/˜arvind/extract/

A in Fig. 11 on the document in Fig. 9(b), we may obtain the final results in Fig. 11. DIs conforming to s_4 in Fig. 7 are returned. Such XQuery scripts can be treated as a representation of extraction rules.

7. Experiments

We conducted experiments to compare our methods with other famous systems. Clustering results based on structural information are compared with a traditional clustering method implemented in CLUTO (A Software Package for Clustering High-Dimensional Data Sets) (Karypis 2002; Steinbach et al. 2000) that does not consider structure of documents. We compare our classification results with Rainbow from CMU[1]. Comparison shows that our methods are better on semistructured document sets. The performance of Algorithm 1 is compared with two recent tree structure miners. We show that our methods deliver quite well extracted results by adjusting various parameters.

7.1. Data sets

Most of the data sets used in our experiments were taken from previous literature, listed in Table 1. In these data sets, each document is assigned a category label. To

[1] http://www-2.cs.cmu.edu/˜mccallum/bow/

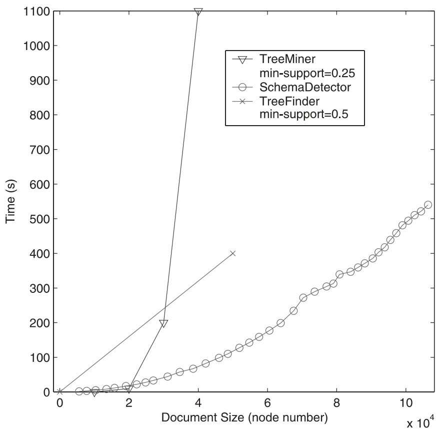


Fig. 12. Performance comparison

evaluate clustering methods, we mix documents from various categories to generate a *MIX* data set. For RoadRunner, we randomly choose 4 documents from each of its 12 categories. We randomly choose 5 documents from each of the 6 categories in RISE (Repository of Online Information Sources Used in Information Extraction Tasks), 4 documents from each of the 10 categories in WIEN. For ExAlg, they provided 3 categories, 132 documents except some documents from the RoadRunner. We chose 50 documents from 3 categories of ExAlg. Most documents in these data sets have regular structures; we also generated the *ETC* (Equivalent Tree Class) data set that is more complicated than the above data sets. Web pages in the ETC data set were crawled from Amazon. These pages have been classified by Amazon into categories; Thus, we can easily verify clustering and classification results. To generate ETC data sets, we only follow hyperlinks in the category list on the left side of the homepage of Amazon. Each hyperlink links to a homepage of a category of commodities. We randomly choose 10 hyperlinks in this area, and in the homepage of each category, we downloaded 3–4 pages following hyperlinks in the navigation area. All these downloaded pages were transformed to the XHTML format using HTMLTidy toolkit. The average size of these documents is 550 K bytes.

7.2. Performance of schema detection

For Web data extraction, it is important to process large-scale documents quickly. For example, if users want to trace price changes in an e-commerce Web site, a long refresh period is not acceptable. We evaluated the complexity of SchemaDetector on the ETC data set. We conducted an experiment on a PIII933 laptop with 512 M memory. SchemaDetector is the key process in our system that finds schema. TreeFinder and TreeMiner are two famous tree-structure miner algorithms. In Fig. 12, we find the complexity results of these three algorithms. The results of TreeFinder and TreeMiner were collected from related papers (Termier et al. 2002; Zaki et al. 2003). The document size of TreeMiner is measured by the string length of documents, not measured by node number. We plot the point of TreeMiner there, because in an ETC data set, the string length of documents containing 50 K nodes is about 1 M, which

Table 2. Clustering accuracy

Data set	E1	E2	W	R1	R2	Average
Our method(%)	90	100	100	100	89.6	95.7
CLUTO(%)	47.5	100	72.5	66.7	70.8	71.2

Table 3. Classification accuracy

Data set	E1	E2	W	R1	R2	Average
Our method(%)	97.5	100	100	100	100	99.5
Rainbow(%)	77.08	86.78	89.58	89.58	87.66	86.14

is approximately equal to the size of the data set used by TreeMiner. TreeMiner and TreeFinder will discard structures that appear fewer times than a minimum support value; when the value decreases, the complexity of them increases rapidly. It is easy to see from Fig. 12 that our algorithm outperformed them when the minimum support was set to a small value in TreeFinder (5%) and TreeMiner (0.25%). We have analyzed that, when the number of schemata is much smaller than the number of subtrees, the complexity of SchemaDetector is near linear; otherwise, its worst complexity is $O(n^2)$. From the diagram, we can see the empirical results verify that. When the data set is small, the number of schemata is large compared with the number of DIs; thus, the complexity grows quickly. With the increase in size of the data set, the complexity is near linear, as the number of schemata is smaller compared with the number of DIs.

7.3. Clustering accuracy

Given the similarity between each pair of documents, we choose the bisecting K-means method to cluster them. Table 2 lists comparison results obtained on MIX data sets, including documents selected from several data sets. Row 1 indicates these data sets: E1 (ETC), E2 (ExAlg), W (WIEN), R1 (RISE), R2 (RoadRunner). CLUTO also uses a bisecting K-means method but only considers texts in documents. CLUTO is very suitable to cluster- free text documents (Steinbach et al. 2000). From this table, we can see that, except for documents from ExAlg, our method was much better than CLUTO. The reason is that CLUTO only performed well when documents contained long free-text paragraphs, e.g. documents in ExAlg. However, our method delivered good results consistently.

7.4. Classification accuracy

Table 3 lists the comparison results of Rainbow and our classification method. We exploited a fourfold cross-validation strategy to evaluate Rainbow; i.e. divided each data set into 4 parts, each time we chose 3 of them as training sets and 1 as a test set. The final result is the average accuracy of results in four times. The accuracy results obtained by Rainbow vary from 77.08% to 89.58%. Our method classifies

Table 4. Extraction accuracy

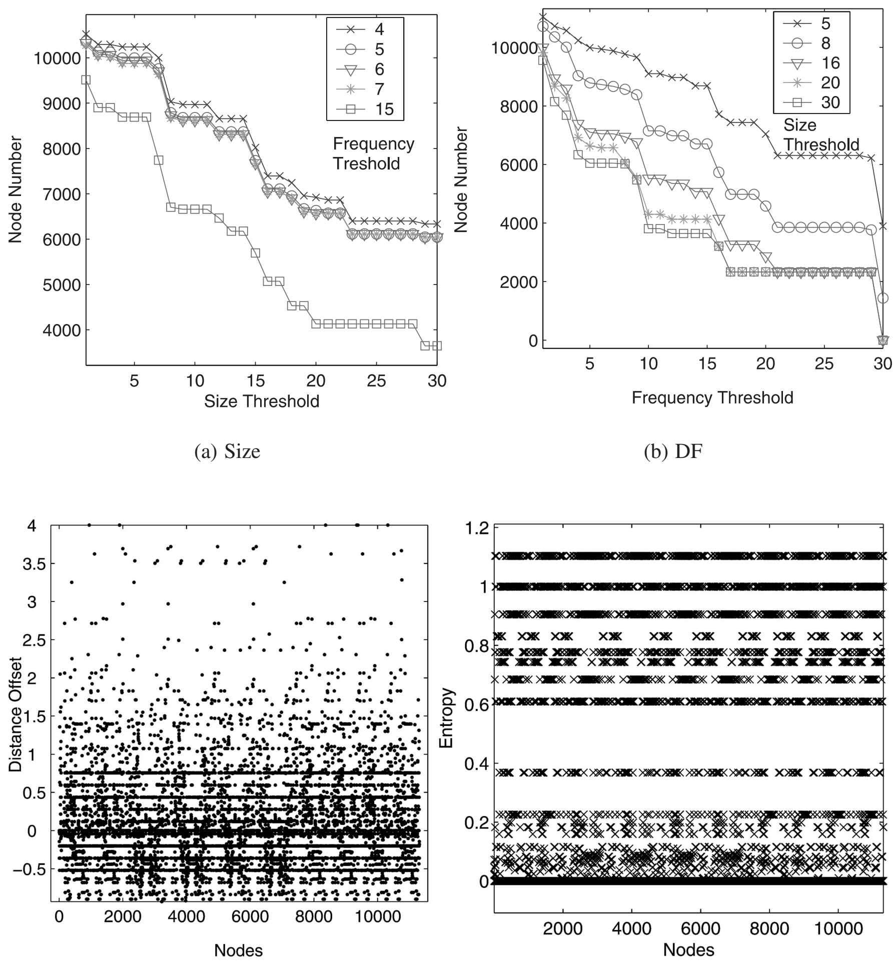
Data set	RecNum	Extracted	Accuracy
Alta	100	90	0.9
Cora	100	70	0.7
Excite	100	100	1
Galaxy	200	199	1
Hotbot	100	95	0.95
L.A. Weekly	81	76	0.94
Magellan	100	92	0.92
Metacrawler	200	179	0.9
Northenlight	100	97	0.97
Openfind	200	185	0.93
SavvySearch	150	140	0.93
StptCom	100	97	0.97
Webcrawler	250	141	0.56
Total	1781	1561	0.88

documents based on Formula 10. It delivers 100% accuracy on all data sets except ETC, and the average accuracy of our method is about 13% higher than Rainbow. On the ETC data set including documents with complicated structures, Rainbow's results are not good, although in other data sets, it can achieve accuracy larger than 85%. Our method is almost not affected by the difference among data sets. On the *ETC* data set, one page was classified to the error class by our methods. The reason is that the page is a page linking other subcategories and it is difficult to judge to which class it belongs.

7.5. Extraction accuracy

We evaluated our extraction method on an IEPAD data set including Web documents returned by 10 search engines listed in the first column of Table 4. We first manually annotated DIs in these documents; the method is quite straightforward; i.e. each entry returned by those searches is treated as a DI. The number of hand-annotated DIs (RecNum) are listed in the second column in Table 4.

We executed SchemaDectector to assign schema ID to each DI in document trees parsed from documents of IEPAD. As a result, most annotated DIs from the same search engine are assigned with the same schema ID. In Table 4, the second column (Extracted) is the number of annotated DIs belonging to the type that contains the largest number of annotated DIs. We observe that our method accurately identifies these important contents in documents. The accuracy values in column 3 are the rate of Extracted to RecNum. For document trees returned by most search engines, our accuracy rate is greater than 90%. The reason for receiving poor accuracy results in documents from Cora and WebCrawler is that these search engines highlight some of their search results using various formats. If we consider the type that contains the second largest set of DIs, the accuracy values will exceed 90%, too.



(a) Size

(b) DF

(c) Offset

(d) Entropy

Fig. 13. Distribution of schema attributes

7.6. Discussion on schema attributes

As introduced before, we justify parameters to select which types of DIs should be extracted. Figure 13(a) shows the relation between size threshold and the number of nodes extracted; i.e. only those DIs conforming to schema with size larger than the threshold are extracted. As the threshold increases, the number of extracted nodes decreases. In some ranges, the number changes very slowly. For example, when the size threshold changes from 25 to 28, the number of extracted nodes almost does not change because there are many DIs belonging to the same schema that has 28 nodes. In Fig. 13(b), when we change the DF threshold from 0 to 30, the number of extracted nodes changes as shown in Fig. 13(a). In Fig. 13(c), we plot the distance offset of each DI. The X-axis is the position of their root nodes in traversal and the Y-axis corresponds to distance offset. Most offsets are located from −1 to 1.5. Empirically, those DIs with large offset are noise. As introduced before, high entropy also means noise sometimes. Figure 13(d) shows that DIs conforming to schemata with low entropy are very easily distinguished from those with high entropy. We may control extraction results by combining the parameters. For example, in the AltaVista documents of *IEPAD*, when we set the size threshold to 6 and frequency threshold to 89, most child DIs of the root node in extracted XML document are those DIs we annotated in the original documents. Figures 13(a) and (b) show the effect of combining two parameters.

8. Conclusion

This paper presents a comprehensive framework for a Web data-extraction system that provides a consistent view over various operations in Web data extraction. Operations, including document-set preparation, document classification and data extraction, are all conducted in schema-based representation of Web documents. To support these operations, similarity measurements for semistructured documents based on schema are proposed. We have also introduced an efficient algorithm to discover frequent structures to generate schema in Web documents. In our experiments on real-world data sets, compared with the methods that do not consider document structures, much better clustering results were achieved using schema-based representation in terms of clustering accuracy. Moreover, we have also demonstrated better document classification results using schema-based representation. Based on the notions introduced in this paper, the Web data-extraction process can be more configurable. We presented how to extend XQuery to extract data fulfilling the requirements of those criteria defined in this paper. In our experiments, promising Web data-extraction results were achieved. Currently, our framework only considers the structure of Web documents. How to combine semantic information in documents to further improve our framework is to be studied. As our methods need to parse documents into trees in memory, it is not efficient on very large data sets. How to solve this problem is our future work. How to adopt other structural methods of classification and clustering in our framework is also to be studied.

References

Akutsu T (1992) An RNC algorithm for finding a largest common subtree of two trees. IEICE Trans Inf Syst E75-D:95–101

Arasu A, Garcia-Molina H (2003) Extracting structured data from web pp. In: Proceedings of SIGMOD conference 2003, pp 337–348

Baumgartner R, Flesca S, Gottlob G (2001) Visual Web information extraction with lixto. In: Proceedings of 27th international conference on VLDB, pp 119–128

Chang C-H, Lui S-C (2001) IEPAD: information extraction based on pattern discovery. In: Proceedings of the 10th international WWW conference, pp 681–688

Crescenzi V, Mecca G, Merialdo P (2001) Roadrunner: towards automatic data extraction from large web sites. In: Proceedings of 27th international conference on VLDB, pp 109–118

Flesca S, Manco G, Masciari E, Pontieri L, Pugliese A (2002) Detecting structural similarities between XML documents. In: Proceedings of 5th international workshop on the Web and databases

Gottlob G, Koch C (2000) Monadic datalog and the expressive power of languages for web information extraction. In: Proceedings of the 21st PODS, pp 17–28

Gupta A, Harinarayan V, Rajaraman A (1998) Virtual database technology. In: Proceedings of the 14th international conference on data engineering, pp 297–301

Karypis G (2002) A clustering toolkit. Technical report TR#2-017, Univ Minnesota

Kosala R, Bruynooghe M, Blokceel H, Van den Bussche J (2003) Information extraction from web documents based on local unranked tree automaton inference. In: Proceedings of the 18th IJCAI-2003

Kushmerick N (2000) Wrapper induction: efficiency and expressiveness. Artif Intell 118(1–2):15–68

Kushmerick N, Thomas B (2002) Adaptive information extraction: core technologies for information agents. In: Intelligent information agents R&D in Europe: an agentlink perspective

Lin S-H, Ho J-M (2002) Discovering informative content blocks from web documents. In: Proceedings of SIGKDD

Liu Z, Li F, Ng WK (2002) Wiccap data model: mapping physical websites to logical views. In: Proceedings of the 21st international conference on conceptual modelling (ER2002)

Nierman A, Jagadish HV (2002) Evaluating structural similarity in XML documents. In: Proceedings of 5th international workshop on the web and databases

Rajaraman A, Ullman JD (2001) Querying websites using compact skeletons. In: Proceedings of PODS

Sakamoto H, Murakami Y, Arimura H, Arikawa S (2001) Extracting partial structures from html documents. In: 14th international Florida artificial intelligence research symposium (FLAIRS'2001) conference, pp 264–268

Steinbach M, Karypis G, Kumar V (2000) A comparison of document clustering techniques. In: Proceedings of KDD workshop on text mining

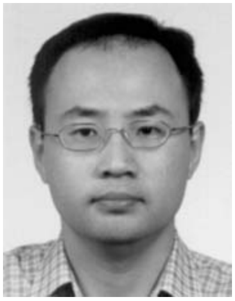
Termier A, Rousset M-C, Sebag M (2002) Treefinder: a first step towards SML data mining. In: Proceedings of IEEE ICDM

Lian W, Cheung DW-L (2004) An efficient and scalable algorithm for clustering XML documents by structure. IEEE Trans Knowl Data Eng 16(1):82–96

Zaki MJ, Aggarwal CC (2003) Xrules: an effective structural classifier for XML data. In: Proceedings of SIGKDD 03

Author biographies

Zhao Li received his MS in computer technique application from Southeast University, Nanjing, China (2001). In 2003, he was admitted into Nanyang Technological University, Singapore, where he is a PhD student in the School of Computer Engineering. In 2001 and 2003, he was working at Huawei Technologies, Beijing, as a research engineer. His research interests include Web data extraction and machine learning.



Wee-Keong Ng (Nanyang Technological University) is an Associate Professor with the School of Computer Engineering and is Director of the Centre for Advanced Information Systems at the Nanyang Technological University, Singapore. He received his PhD from the University of Michigan at Ann Arbor, USA, in 1996. Dr. Ng has published more than 100 refereed journal and conference articles in the area of data mining, web information systems, database and data warehousing and software agents. His papers have appeared in the IEEE Transactions on Knowledge and Data Engineering, ACM Computing Surveys, Data and Knowledge Engineering Journal, World Wide Web Journal, and other major journals. Dr. Ng actively participates in international conference activities. He serves in the program committees of numerous international conferences annually and has held tutorials and talks at conferences and research seminars. Dr. Ng is a member of IEEE Computer Society and a member of ACM.

Aixin Sun received his PhD and BASc with first-class honors from the Nanyang Technological University (Singapore) in 2004 and 2001 respectively, both in computer engineering. He has been a research engineer with the Center for Research in Pedagogy and Practice (CRPP), National Institute of Education, Singapore. His research interests include text/Web classification, digital libraries and information extraction.

Correspondence and offprint requests to: Zhao Li, Centre for Advanced Information Systems, School of Computer Engineering, Nanyang Technological University, Nanyang Avenue, Singapore 639798.
Email: liz@pmail.ntu.edu.sg