OntoMiner: automated metadata and instance mining from news websites

Hasan Davulcu and Srinivas Vadrevu*

Department of Computer Science and Engineering
Arizona State University, Tempe, AZ 85287-8809, USA
E-mail: hdavulcu@asu.edu
E-mail: svadrevu@asu.edu
*Corresponding author

Saravanakumar Nagarajan

Convera Corporation
1808 Aston Avenue
Carlsbad, CA-92008, USA
E-mail: snagarajan@convera.com

Abstract: RDF/XML has been widely recognised as the standard for annotating online web documents and for transforming the HTML web into the so-called Semantic Web. In order to enable widespread usability of the Semantic Web, there is a need to bootstrap large, rich and up-to-date domain ontologies that organise the most relevant concepts, their relationships and instances. In this paper, we present automated techniques for bootstrapping and populating specialised domain ontologies by organising and mining a set of relevant overlapping websites. We develop algorithms that detect and utilise HTML regularities in the web documents to turn them into hierarchical semantic structures encoded as XML. Next, we present tree-mining algorithms that identify key domain concepts and their taxonomical relationships. We also extract semi-structured concept instances annotated with their labels whenever they are available. We also report experimental evaluation for the news, travel and shopping domains to demonstrate the efficacy of our algorithms.

Keywords: metadata; instance ontology; mining; news; automation; web; semantic.

Reference to this paper should be made as follows: Davulcu, H., Vadrevu, S. and Nagarajan, S. (2005) ‘OntoMiner: automated metadata and instance mining from news websites’, *Int. J. Web and Grid Services*, Vol. 1, No. 2, pp.196–221.

Biographical notes: Dr. Hasan Davulcu is Assistant Professor in the Department of Computer Science and Engineering at Arizona State University (ASU). He received his PhD degree from University at Stony Brook, NY in 2002. He founded the Cognitive Information Processing Systems Lab at ASU. His research interests include ontology-directed information extraction, the semantic web, data and text mining, and information integration.

Srinivas Vadrevu is a PhD student in the Computer Science and Engineering Department at Arizona State University. Prior to that, he received the MS degree in Computer Science from the University of Minnesota Duluth, and BTech degree from Nagarjuna University, India. His research interests include web mining, machine learning, ontologies and artificial intelligence.

Saravanakumar Nagarajan graduated with master's degree in the Computer Science and Engineering Department at Arizona State University and is currently working in Convera. His research interests include semantic web mining, information retrieval and web spam detection. He received his bachelor's degree in Computer Science from the University of Madras, India.

1 Introduction

RDF and XML have been widely recognised as the standard for annotating online web documents and for transforming the HTML web into the so-called Semantic Web. Several researchers have recently questioned whether participation in the Semantic Web is too difficult for 'ordinary' people (McDowell *et al.*, 2003; McBride, 2002). In order to enable widespread usability of the Semantic Web there is a need to bootstrap large, rich and up-to-date domain ontologies that organise most relevant concepts, their relationships and instances. In this paper, we present automated techniques for bootstrapping and populating specialised domain ontologies by organising and mining a set of relevant overlapping taxonomy-directed websites. A website is said to be 'taxonomy-directed' if it contains at least one taxonomy for organising its key concepts and it presents the instances belonging to each concept in a regular fashion. Notice that, neither the presentation of the taxonomy among different pages, nor the presentation of instances for different concepts, needs to be regular for a website to be classified as 'taxonomy directed'. Almost all scientific news, financial, travel, shopping and search/community portals that we are aware of are indeed 'taxonomy directed'. A pair of websites are said to be *overlapping* if their taxonomies share some concept labels.

The user of the OntoMiner system only needs to provide the system with a list of Home Pages of taxonomy-directed overlapping websites that characterise her domain of interest. Collections of such websites can be obtained from rich and diverse online directories such as Yahoo![1] or the Open Directory Project.[2] Next, the OntoMiner system detects and utilises the HTML regularities within every web document and turns them into hierarchical semantic structures encoded as XML by utilising a hierarchical partition algorithm. We present tree-mining algorithms that identify most important key domain concepts selected from within the directories of the Home Pages. OntoMiner proceeds with expanding the mined concept taxonomy with subconcepts by selectively crawling through the links corresponding to key concepts. OntoMiner also has algorithms that can identify the logical regions within web documents that contain links to instances. OntoMiner can accurately separate the 'human-oriented decoration', such as navigational panels and advertisement bars, from relevant information and utilises the inferred hierarchical partition corresponding to relevant segments to accurately collect the semi-structured concept instances.

A key characteristic of OntoMiner is that, unlike the early pioneering systems described in Crescenzi *et al.* (2001), Arasu and Garcia-Molina (2003) and Chung *et al.* (2002), which attempt to extract data values alone, OntoMiner extracts the labels corresponding to categories and attribute names along with their data values and organises them into an ontology. We formally define an *ontology* to be a seven-tuple:

$$O = < L, C, T, I, A, Syn, Member >$$

where:

L – is a set of labels denoting concepts and attributes

C – is a set of concepts

T – is a taxonomy relationship among concepts, $T \subseteq C \times C$, which is reflexive, asymmetric and transitive

I – is a set of instances

A – is a set of attributes, where each attribute is a function that maps instances to labels, $A: I \rightarrow 2^L$

Syn – is a function that maps concepts and attributes to their various labels, $Syn: \{C \cup A\} \rightarrow 2^L$

$Member$ – is a function that represents member-of relationship between instances and concepts, $Member: I \rightarrow C$.

As an example application, a user of the OntoMiner can use the system to rapidly bootstrap an ontology populated with instances, and the user can tidy up (Doan *et al.*, 2003; Madhavan *et al.*, 2002; Noy and Musen, 2000) the bootstrapped ontology to create a rich set of labelled examples that can be utilised by supervised machine learning systems such as the WebKB (Craven *et al.*, 1998). Also, OntoMiner can separate the data instances from the data labels within the vicinity of extracted data and attempts to accurately annotate the extracted data by using these labels whenever they are available. Related work includes schema learning (Nestorov *et al.*, 1998; Garofalakis *et al.*, 2000; Papakonstantinou and Vianu, 2000) for semi-structured data and techniques for finding frequent substructures from hierarchical semi-structured data (Cong *et al.*, 2002; Zaki, 2002). These techniques can be utilised to train structure-based classifiers (Theobald *et al.*, 2003; Bertino *et al.*, 1999) to help merge and map between similar concepts of the bootstrapped ontologies and better integrate their instances.

The rest of the paper is organised as follows. Section 2 outlines related work and distinguishes our contributions. Section 3 presents a motivating example and an overview of the system and its architecture. Section 4 presents the semantic partitioning algorithms, in particular the flat and hierarchical partitioning algorithms for web pages. Sections 5 and 6 discuss taxonomy mining and instance extraction algorithms. Section 7 presents the experimental results for news, travel and shopping domains. We discuss future work in Section 8.

2 Related work

Information extraction from the web is a well-studied problem and related work can be categorised as wrapper development tools, semi-automated wrapper learning, template-based automated algorithms, ontology-based approaches, grammar induction-based algorithms, and hierarchical structuring-based approaches.

2.1 Wrapper development tools

Wrappers (Hammer *et al.*, 1997; Arocena and Mendelzon, 1998) are scripts that are created either manually or semi-automatically after analysing the location of the data in the HTML pages. Wrappers tend to be brittle against variations and require maintenance when the underlying websites change.

2.2 Semi-automated wrapper learning

Wrapper induction systems (Kushmerick *et al.*, 1997; Muslea *et al.*, 1999) generate extraction rules from a given set of labelled training examples. These extraction rules can be applied on other new pages to extract the data. These systems utilise the natural language processing techniques and HTML tags to infer extraction patterns. Our OntoMiner approach does not require any training examples when performing extraction.

2.3 Template-based algorithms

RoadRunner (Crescenzi *et al.*, 2001) works with two documents from a collection of template-generated web pages to infer a template for the collection using union-free regular expressions. ExAlg (Arasu and Garcia-Molina, 2003) is another system that can extract data from template-generated web pages. ExAlg uses equivalence classes (sets of items that occur with the same frequency in every page) to build the template for the pages by recursively constructing the page template starting from the root equivalence class. Table segmentation work presented in Lerman *et al.* (2004) proposes methods to efficiently segment HTML tables and lists into records by mining the page templates. Our work differs from such approaches since it performs extraction without the assumption that the object instance pages should be template driven.

2.4 Ontology-based approaches

Ontology-based approaches (Embley *et al.*, 1998; Dill *et al.*, 2003; Hong and Clark, 2001) require fully developed domain ontology and utilise matching and disambiguation algorithms to locate and extract the entities and relationships of interest. On the other hand, the *KnowItAll* system (Etzioni *et al.*, 2004) requires only a set of class and relation names with a small set of generic rule templates with an assessor algorithm to extract and rank facts from free-text segments. Our OntoMiner approach is domain independent and does not require a previously engineered domain ontology.

2.5 Grammar induction-based algorithms

Grammar induction-based systems employ a strong bias on the type and expected presentation of items within web pages to extract instances. Such assumptions like ‘product descriptions should reside on a single line’ (Doorenbos *et al.*, 1997) and ‘items may not have missing or repeating attributes’ (Doorenbos *et al.*, 1997; Yang *et al.*, 2001) do not apply for most of the websites.

2.6 Hierarchical structuring-based approaches

The approaches proposed by Yang *et al.* (Yang and Zhang, 2001) and Gertz *et al.* (Chung *et al.*, 2002) are similar to our approach. However, Yang (Yang and Zhang, 2001) views the HTML as a sequence of HTML tags and text and builds semantic structures by detecting maximal patterns that identify the boundaries between various logical sections of a web page. They utilise a distance function which measures the similarity between various sequences of HTML tag paths to discover the recurring patterns in the document. This similarity function must be hand-coded for different web pages for various domains of interest. Also, they do not address the problem of labelling the semantic groups in the document. Their goal is similar to ours; however, they focus on topic-specific HTML documents from a domain of interest, whereas our approach does not require any tuning or domain-specific knowledge. Another category of ontology mining work (Riloff and Shepherd, 1997; Maedche and Staab, 2000; Navigli *et al.*, 2003) uses free-text corpus and various natural language processing and data mining techniques to bootstrap ontologies. However, the OntoMiner system utilises only presentation regularities of semi-structured data embedded within web documents to organise them into a partial bootstrapped ontology. Our system detects free-text HTML leaf nodes and does not process them any further.

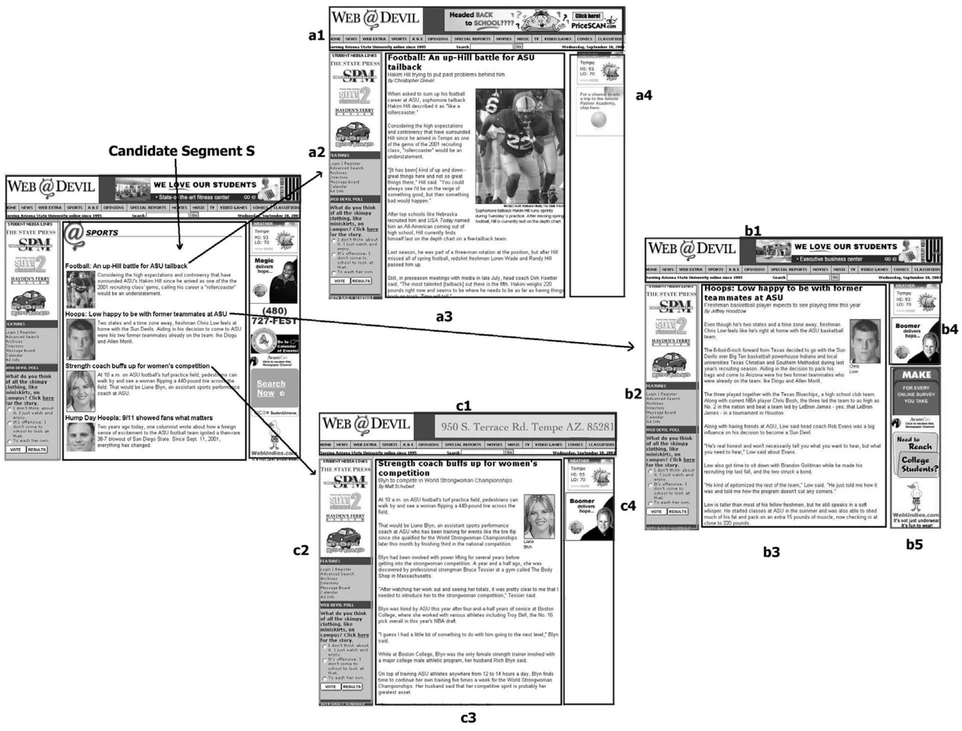
3 Motivating example and architecture

In this section we provide a motivating example for the OntoMiner system and illustrate the architecture of the system.

3.1 Motivating example

To illustrate a simple example, consider the four web pages shown in Figure 1. The web page on the left is the home page of a news website, and the three web pages on the right are the individual instance pages that point to individual news articles. The home page of the news website is 'taxonomy-directed', as it contains a taxonomy of news categories such as *World*, *USA*, *Weather*, *etc.* on the left. Given a collection of such home pages, the OntoMiner system can extract the taxonomies from individual websites, merge the taxonomies to provide a common taxonomy, and identify and extract instances for each of the categories in the taxonomy. For example, the OntoMiner is able to extract the instances for the 'Headline News' category in the taxonomy from the three web pages shown on the right in Figure 1. The segments in the solid boxes of the three web pages on the right show the individual instance information (which in this case are news articles).

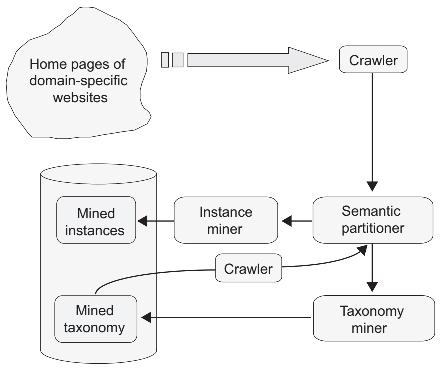
Figure 1 Motivating example with illustration of the template of instances from a concept web page. Solid boxes denote the content of the instances among instance pages and dashes boxes denote mismatch segments that may correspond to 'human-oriented decoration'.


Candidate Segment S
a1
a2
a3
a4
b1
b2
b3
b4
b5
c1
c2
c3
c4
Football: An up-Hill battle for ASU tailback
Hoops: Low happy to be with former teammates at ASU
Strength coach buffs up for women's competition
Hump Day Hoopla: 9/11 showed fans what matters

3.2 OntoMiner architecture

The architecture of OntoMiner is shown in Figure 2. As the Crawler fetches the web pages, they are fed to the Semantic Partitioner. The Semantic Partitioner examines the structure of the HTML page and converts the page's contents into a labelled hierarchy by utilising regularities in the visual presentation of the HTML page. The Taxonomy Miner examines these hierarchies and mines the taxonomy of concepts. For each concept in the mined taxonomy, the Instance Miner identifies the concept instances. The following sections describe each of these modules in detail. The OntoMiner algorithm is detailed in Algorithm 1 of Figure 2. The input to the OntoMiner algorithm is a set of Home Pages and an experimentally determined support value. Its output is a bootstrapped ontology. In Line 1, it semantically partitions the collection of Home Pages, using the algorithm discussed in Section 4. Next, for these semantically partitioned pages, it invokes the Taxonomy Miner algorithm, which is discussed in Section 5. The Taxonomy Miner algorithm takes the semantically partitioned pages and the experimentally determined support value as inputs, and mines and populates the ontology. In Line 1, the Taxonomy Miner finds the frequent labels, *L*, and groups them into concepts, *C*, as described in Section 5.1. Next, in Line 2, it invokes the instance extraction algorithm, described in Section 6, to find and extract the instances, *I*. It also identifies their set of attribute-value pairs and records them in *A*. The membership relations between instances and concepts are recorded using the *Member* function. In Line 3, the algorithm uses the TaxMiner, described in Section 5.4, to organise the concepts in *C* into a taxonomy *T*. Next, in Lines 5–9, it retrieves the next batch of pages corresponding to the *URLs* of the labels of mined concepts and invokes the OntoMiner algorithm recursively on this new set of pages to expand the domain ontology.

Figure 2 Architecture and algorithm of OntoMiner



Algorithm 1 Algorithm to bootstrap ontologies from home pages of overlapping domain-specific websites

```
OntoMiner
Inputs: H: Set of Pages, O: Ontology, Sup: Support
Output: O: Ontology
  1: $Sem_H$ = Semantic-Partitioner(H)
  2: O = Taxonomy-Miner($Sem_H$, Sup)
End of OntoMiner

Taxonomy-Miner
Inputs: $Sem_H$: Set of Semantically Partitioned Pages, Sup: Support
Output: O: Ontology
  1: < $\mathcal{L}$, C, Syn > = Frequent-Label-Mining ($Sem_H$, Sup)
  2: < $\mathcal{A}$, $\mathcal{T}$, Member > = Instance-Extraction($Sem_H$, C, Sup)
  3: $\mathcal{T}$ = TaxMiner(C, $Sem_H$, Sup)
  4: O = < L, C, T, I, A, Syn, Member >
  5: if Syn(C), url != ø then
  6:     H’ = fetch(Syn(C).url)
  7:     OntoMiner(H, O, Sup)
  8:  else
  9:      Return O
  10:  end if
```

4 Semantic partitioning

OntoMiner employs a three-phase semantic partitioning algorithm made up of flat partitioning, data table detection and hierarchical partitioning.

4.1 *Flat partitioning*

Our Flat Partitioner (FP) algorithm detects various logical segments within a web page. For example, for the home page of www.nytimes.com, we marked the logical segments in boxes B1 through B5 in Figure 3a.

The boundaries of Segments B2 and B3 correspond to the dotted lines shown in the DOM tree of the web page in Figure 3b. Intuitively, FP groups similar contiguous substructures in the web pages into logical segments by detecting a high concentration of neighbouring nodes with similar root-to-leaf tag paths. First, FP initialises the segment boundary to be the first leaf node of the DOM tree. Next, it connects a pair of leaf nodes with a ‘similarity-link’ if they share the same root-to-leaf path and if all other leaf nodes in between have different paths. Then, it calculates the ratio of cardinality of similarity-links crossing the current candidate boundary to that of those within the current segment. If this ratio is less than an experimentally determined threshold δ, set to 0.34, then FP marks the current node as a segment boundary. Otherwise, it adds the current node to the current segment and considers the next node as a segment boundary. The process terminates when the algorithm reaches the last leaf node. The tree view in Figure 3b illustrates the FP algorithm listed in Algorithm 2 of Figure 4.

Figure 3 Snapshot of New York Times home page

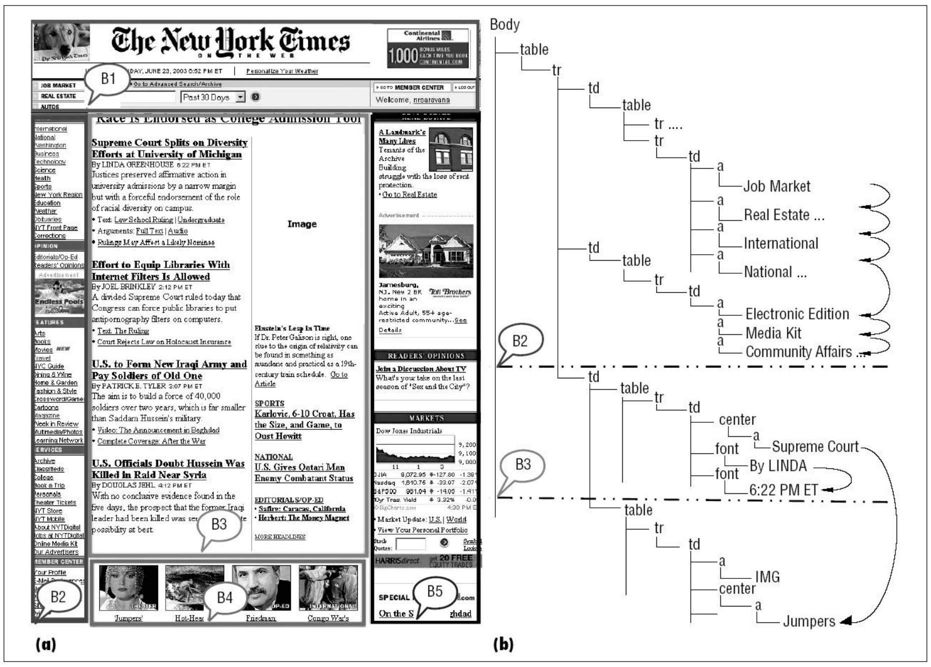


Figure 4 Flat partitioner algorithm and cost factors for hierarchical partitioning

```
Algorithm 2 Flat Partition Algorithm
Flat Partitioner
Input: T: DOM Tree
Output: $\{b_1, b_2, ...b_k\}$: Flat Boundaries, set of leaf nodes

 1: PIT := PathIndexTree(T)
 2: current := first leaf node of $T$
 3: PartitionNodes := $\phi$
 4: SegmentBoundaries := $\phi$
 5: for each lNode in LeafNodes(T) do
 6:     current := lNode $\rightarrow$ next
 7:     if n = PIT.nextSimilar(current) exists then
 8:         PartitionNodes := PartitionNodes $\bigcup$ {n}
 9:     end if
10:     out = $|\{path(m)|m \in PartitionNodes\ and\ m > current\}|$
11:     total = $|\{path(m)|m \in PartitionNodes\}|$
12:     if out/total $< \delta$ then
13:         SegmentBoundaries := SegmentBoundaries $\bigcup$ {current}
14:         PartitionNodes := $\phi$
15:     end if
16: end for
17: Return SegmentBoundaries
```

1. $C_{LCA}(A,B) = \sqrt{\frac{d_1+d_2}{2*max\_depth}}$

2. $C_{PSIM}(A,B) = 1 - \frac{Similarity\ between\ Paths\ P_1\ and\ P_2}{max(d_1,d_2)}$

3. $C_{STSIM}(A,B) = 1 - max(Separation, Overlap)$,
where $Separation = \frac{|\{S_1-S_2\}\bigcup\{S_2-S_1\}|}{|S_1\bigcup S_2|}$
and $Overlap = \frac{|S_1\bigcap S_2|}{|S_1\bigcup S_2|}$

4. $C_{ORD}(A,B) = 1 - Sim(A,B)$,
where Sim(A,B) =

$$\frac{Number\ of\ Similar\ Paths\ in\ order\ in\ Sub\ Trees\ of\ A\ and\ B}{Max\ of\ No\ of\ Paths\ in\ Sub\ Trees\ of\ A\ and\ B}$$

5. $C_{NSIM}(A,B) = 0.5 * depth\_ratio + 0.5 * node\_ratio$,

where $depth\_ratio = \frac{min\_depth}{max\_depth}$,
$min\_depth = min(depth\ of\ A,\ depth\ of\ B)$
$max\_depth = max(depth\ of\ A,\ depth\ of\ B)$

$node\_ratio = \frac{min\_nodes}{max\_nodes}$,
where $min\_nodes = min(|Nodes\ in\ A|,\ |Nodes\ in\ B|)$
$max\_nodes = max(|Nodes\ in\ A|,\ |Nodes\ in\ B|)$

4.2 Data table detection and extraction

Tables are an important means for communicating important information, and understanding such tables is a challenging problem in document layout analysis. For example, the web page shown in Figure 3a contains a data table containing stock market information. In order to effectively retrieve the attribute labels and their values from such tables, an algorithm to accurately detect and extract data from tables is required. Over the past decade, many techniques (Hurst, 2002; Wang and Hu, 2002; Chen *et al.*, 2002) have been proposed for data table detection that are based on training a classifier with large sets of data tables and testing it on other tables. Many of these techniques are not completely unsupervised and hence not applicable to the Semantic Partitioner. This section discusses a novel unsupervised approach that uses HTML regularities for detecting *Data* tables and a clustering algorithm for identifying the *template* for the *Data* table.

The Data Table Detector and Extractor (DTDE) is a part of the Semantic Partitioner and it has two separate components. The first component is the Data Table Detector (DTD) which identifies all the candidate data tables in the hyper text document. The DTD sends all the candidate data tables to the Data Table Extractor (DTE), which has three subcomponents: Data Table Cleaner (DTC), Data Table Template Identifier (DTTI) and Data Extractor (DE). The DTC modifies the structure of the table based on the spanning of rows and columns. The DTTI uses a special clustering technique to identify the template of the table. The DE extracts data from the table using the template given by the DTTI. Finally, the extracted data along with the template is given to the Semantic Partitioner.

4.2.1 Data Table Detector (DTD)

The DTD identifies data tables from the candidate HTML tables using the following two heuristic rules. A candidate table node in the DOM tree of HTML document, <*table*>, is considered *data* table if:

- The value of the border attribute of the candidate table is greater than zero, or it has at least two rows and two columns.
- The number of rows with the same number of columns in the candidate table is above a certain threshold.
- The cells in the table should cluster into at least two clusters, with a maximum number of cells in each row or column that groups to a cluster that must be above a certain threshold.

The threshold value for the second and third rules has been experimentally determined to be 80%. The first rule assumes that all the tables with a border or with at least two rows and columns in HTML documents are data tables. The next rule states that there should be at least 80% of the rows in the table that have an equal number of columns. Then, the DTD sends all the data table nodes to the Data Table Extractor.

4.2.2 Data Table Extractor (DTE)

The Data Table Extractor has three components: the Data Table Cleaner (DTC), the Data Table Template Identifier (DTTI) and the Data Extractor (DE).

Data Table Cleaner (DTC)

The DTC removes all the cells spanning the table, if any exists. This step is necessary to normalise the number of cells in each row of the table. Intuitively, it replicates the cells that span over columns into the corresponding row, and the cells that span over rows into the individual rows. Let us consider the example shown in Table 1, which has a cell labelled *Adult*, with a row span, and another one labelled *Lodge*, with a column span. One can see that *Lodge* spans columns 1 and 2 and that *Adult* spans rows 2, 3 and 4.

Table 1 A sample hotel information table

Lodge		*High class*	*Middle class*	*Low class*
Adult	Single bed	$$	$$	$$
Adult	Double bed	$$	$$	$$
Adult	Extra	$$	$$	$$

After replicating *Lodge* in column 2 of row 1 and *Adult* over rows 2, 3 and 4, Table 2 is obtained.

Table 2 Modified table obtained from data table cleaner

Lodge	*Lodge*	*High class*	*Middle class*	*Low class*
Adult	Single bed	$$	$$	$$
Adult	Double bed	$$	$$	$$
Adult	Extra	$$	$$	$$

Data Table Template Identifier (DTTI)

The DTTI recognises the orientation of the table by identifying its header cells. It uses cell similarities in the table to cluster the header cells into one cluster and nonheader cells into another. Here, the assumption is, 'All header cells share similar features with their corresponding row or column'. Hence, the DTTI does not cluster the whole table into two clusters. Instead, it groups each column (or row) into two clusters of header and nonheader cells. Next, it identifies the header cluster, *i.e.*, the cluster with header cells, from two clusters (in any column) by a simple heuristic rule: 'Always classify the first cell in the table as header cell'. This rule is based on the assumption that in any data table, the first cell is always part of the header. You can observe that in Figure 5, the first cell in all types of orientation of the *data* table is a *header* cell. Hence, it classifies any cell that clusters with the first cell as a header cell. It repeats the clustering process, recursively labelling all the cells that cluster with 'header cells'. The five-step process of template identification is described as follows:

1 Extract features for each cell in the table.

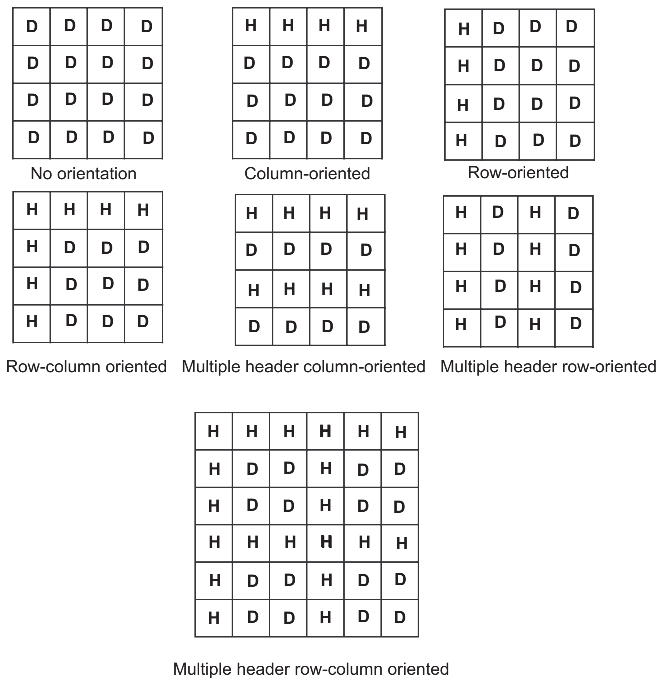
2 Cluster each column (or row) into two clusters.

3 Label the first cell of the table 'header cell'.

4 Classify all the cells in the cluster that are the same as the header cell as header cells.

5 Recursively label all the cells in the table 'header cells' if they cluster with the above set of header cells.

Figure 5 Various orientations of data table


No orientation
Column-oriented
Row-oriented
Row-column oriented
Multiple header column-oriented
Multiple header row-oriented
Multiple header row-column oriented

Extracting features

The following list of features is extracted from each cell in the table as attributes for clustering:

1 Syntactic features like bold, italic, capitalised and hyperlinked.

2 Grammatical features like part of speech.

3 Semantic features like number and date.

4 Statistical features like number of words and word length.

5 Special characters like $, + and {.

All these features are given different weights after analysing a lot of *data* tables over the web.

Clustering

The next step is clustering each column (or row) in the table into two clusters using the features extracted from the previous step. First, the intracluster distances of all the cells in each column (or row) are calculated, and if each is less than an experimentally determined threshold value, then all of the cells are grouped into one cluster. Otherwise, they are separated into two clusters using simple K-Means clustering algorithm. Next, the first cell in the table is classified as a header cell and all the cells that are grouped into a cluster with the first cell are also classified as header cells. The above two steps are repeated by clustering cells in the rows of this set of header cells. Finally, all the cells that are left unmarked are classified as nonheader cells. The above process is based on the observation that any header cell in a table shares some common features with other header cells in its row or column. Also, the first cell in the table is always part of the header, as can be observed in Figure 5.

Template identification

Once the header cells are marked, the template can be identified to be any one of the possible orientations discussed above. For the hotel information example given in Table 1, the marked header cells are shown in Table 3. The orientation (or template) of the table in Table 1 is identified to be *row-column oriented* using Table 3.

Table 3 Identified header cells for hotel information table

Header	*Header*	*Header*	*Header*	*Header*
Header	Header	Data	Data	Data
Header	Header	Data	Data	Data
Header	Header	Data	Data	Data

Data Extractor (DE)

DE extracts the data from the data table using the template identified by the DTTI. For different orientations, the DE extracts data accordingly. For the above hotel information example, whose orientation is multiple header row-columns, the data is extracted to a matrix of the form shown in Table 4. The cell D_{ij} denotes the data corresponding to the attribute pair in the i^{th} row in the first column and the j^{th} column in the first row. Similarly, the DE extracts the data from various other data tables based on the corresponding orientation obtained from the DTTI.

Table 4 Extracted data from hotel information table

X	*High class*	*Middle class*	*Low class*
Lodge.adult.single bed	D_{11}	D_{12}	D_{13}
Lodge.adult.double bed	D_{21}	D_{22}	D_{23}
Lodge.adult.extra	D_{31}	D_{32}	D_{33}

4.3 Hierarchical Partitioning

The Hierarchical Partitioning (HP) algorithm infers hierarchical relationships among the leaf nodes of the DOM tree of an HTML page, where all the document contents are stored. HP achieves this through a sequence of three operations: binary semantic partitioning, grouping and promotion.

4.3.1 Binary hierarchical partitioning

The binary hierarchical partitioning of a web page is based on a dynamic programming algorithm that employs the following cost function. It basically creates a hierarchical binary parenthesisation of all the leaf nodes yielding a binary partition tree. We recursively define the cost for grouping any two nodes in the DOM tree as follows:

$$Cost(L_i, L_j) = \begin{cases} 0 & \text{if } i = j \\ \min_{i \le k < j} \{Cost(L_i, L_k) + Cost(L_{k+1}, L_j) + \\ Grouping\_Cost(L_{i \ldots k}, L_{k+1 \ldots j})\} & \text{if } i < j \end{cases}$$

where L_i, L_j are any two leaf nodes in the DOM tree.

The cost function calculates the measure of dissimilarity between two internal or leaf nodes, that is, a high value of cost indicates that these two nodes' subtrees are highly dissimilar. Thus, the dynamic programming algorithm finds the lowest cost among all possible binary groupings of nodes and parenthesises them into a binary tree, where similar nodes are grouped together. The cost for grouping two consecutive subtrees is calculated as the sum of five cost factors. Let *A* and *B* be the Lowest Common Ancestors (LCA) of nodes L_i to L_k and L_{k+1} to L_j, respectively.

Then,

Grouping Cost($L_{i \ldots k}$,$L_{k+1 \ldots j}$) = Grouping Cost(*A*, *B*) = $C_{LCA}(A, B) + C_{PSIM}(A, B) + C_{STSIM}(A, B) + C_{ORD}(A, B)$.

The first cost factor $C_{LCA}(A, B)$ calculates how far the two nodes are apart from their LCA. The cost for similarity between paths to the LCA is determined by the second cost factor $C_{PSIM}(A, B)$. The third $C_{STSIM}(A, B)$ and fourth $C_{ORD}(A, B)$ cost factors compute the cost for similarity in the subtrees of the two nodes; the former computes the similarity in the paths whereas the latter computes the shared ordering of paths in the subtree. Finally, the fifth cost factor determines how similar the nodes in the subtrees are.

Let S_1 be the set of all paths in the subtree of *A*, S_2 be the set of all paths in the subtree of *B*, d_1 be the number of tags on the path from LCA to *A*, d_2 be the number of tags on the path from LCA to *B* and max depth be the maximum depth of the DOM tree; then the cost factors are defined as shown in Figure 4.

We need to adjust the five cost functions to fit three other special cases that might arise during the cost function evaluation (see Figure 6).

Case 1 The LCA of the two nodes which are to be grouped into one partition *is one of the nodes* itself, and the other node is not a leaf node.

Case 2 The LCA of the two nodes which are to be grouped into one partition *is one of the nodes* itself, and the other node is a leaf node.

Case 3 The LCA nodes for the ranges are identical.

In all these three cases, the second, fourth and fifth cost factors are irrelevant and, hence, they are ignored. For Case 3, the first cost factor is also ignored. Accordingly, the first and third cost factors are modified as follows:

For Case 1:

$$C_{LCA} = \frac{d}{maxdepth}, C_{STSIM} = 1 - \max(Separation, Overlap)$$

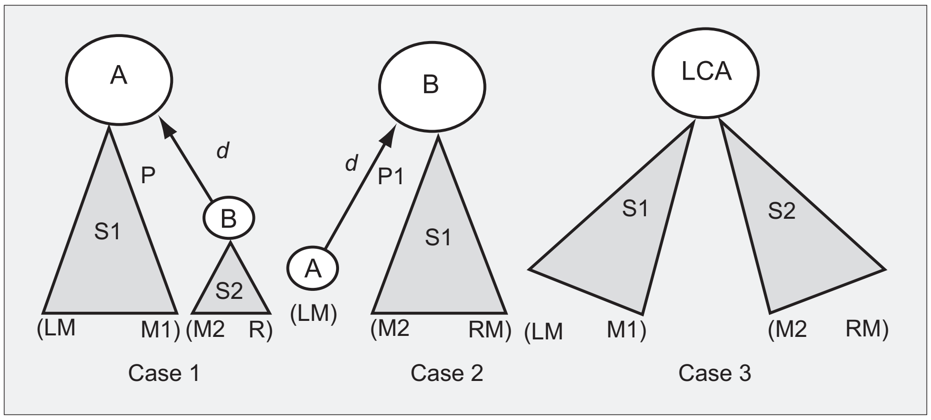
For Case 2, where S_2 is $\{P_1\}$:

$$C_{LCA} = \frac{d}{maxdepth}, C_{STSIM} = 1 - \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$$

For Case 3:

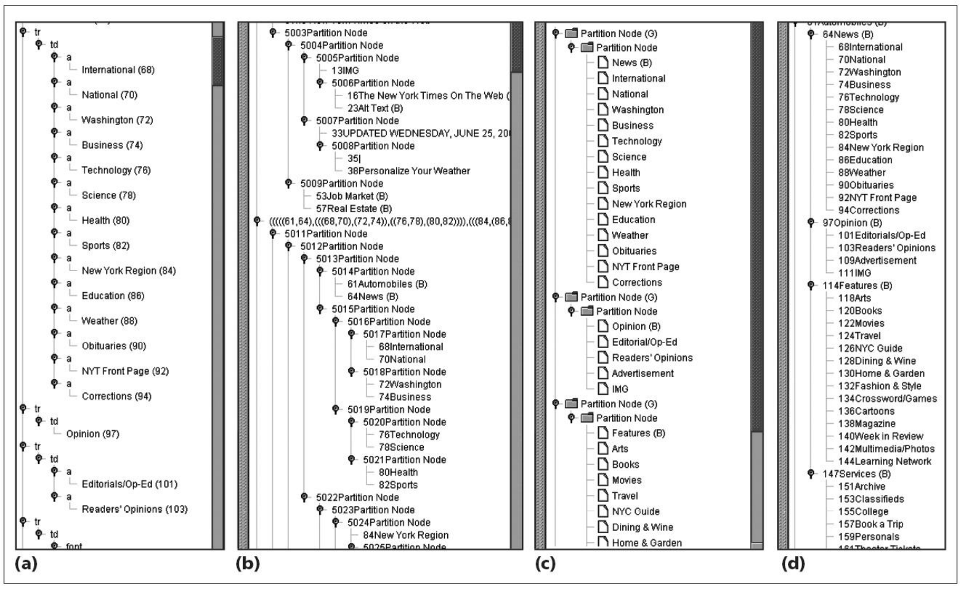
$$C_{STSIM} = 1 - \max(Separation, Overlap).$$

Figure 6 Three different cases during cost function evaluation



In Figure 7, Column 1 (7a) represents part of the DOM tree of the *New York Times* home page, and Column 2 (7b) represents the binary hierarchical partition tree. You can see, for example, that the algorithm groups nodes 68 through 82 into one partition, which has internal binary partitions.

Figure 7 Dynamic programming



```
tr
  td
    a
      International (68)
    a
      National (70)
    a
      Washington (72)
    a
      Business (74)
    a
      Technology (76)
    a
      Science (78)
    a
      Health (80)
    a
      Sports (82)
    a
      New York Region (84)
    a
      Education (86)
    a
      Weather (88)
    a
      Obituaries (90)
    a
      NYT Front Page (92)
    a
      Corrections (94)
tr
  td
    Opinion (97)
tr
  td
    a
      Editorials/Op-Ed (101)
    a
      Readers' Opinions (103)
tr
  td
```

```
    5003Partition Node
      5004Partition Node
        5005Partition Node
          13IMG
          5006Partition Node
            16The New York Times On The Web (
            23Alt Text (B)
        5007Partition Node
          33UPDATED WEDNESDAY, JUNE 25, 20
          5008Partition Node
            35|
            38Personalize Your Weather
      5009Partition Node
        53Job Market (B)
        57Real Estate (B)
  (((((61,64),(((68,70),(72,74)),((76,78),(80,82)))),(((84,(86,
    5011Partition Node
      5012Partition Node
        5013Partition Node
          5014Partition Node
            61Automobiles (B)
            64News (B)
          5015Partition Node
            5016Partition Node
              5017Partition Node
                68International
                70National
              5018Partition Node
                72Washington
                74Business
            5019Partition Node
              5020Partition Node
                76Technology
                78Science
              5021Partition Node
                80Health
                82Sports
        5022Partition Node
          5023Partition Node
            5024Partition Node
              84New York Region
```

```
Partition Node (G)
  Partition Node
    News (B)
    International
    National
    Washington
    Business
    Technology
    Science
    Health
    Sports
    New York Region
    Education
    Weather
    Obituaries
    NYT Front Page
    Corrections
Partition Node (G)
  Partition Node
    Opinion (B)
    Editorial/Op-Ed
    Readers' Opinions
    Advertisement
    IMG
Partition Node (G)
  Partition Node
    Features (B)
    Arts
    Books
    Movies
    Travel
    NYC Guide
    Dining & Wine
    Home & Garden
```

```
  64News (B)
    68International
    70National
    72Washington
    74Business
    76Technology
    78Science
    80Health
    82Sports
    84New York Region
    86Education
    88Weather
    90Obituaries
    92NYT Front Page
    94Corrections
  97Opinion (B)
    101Editorials/Op-Ed
    103Readers' Opinions
    109Advertisement
    111IMG
  114Features (B)
    118Arts
    120Books
    122Movies
    124Travel
    126NYC Guide
    128Dining & Wine
    130Home & Garden
    132Fashion & Style
    134Crossword/Games
    136Cartoons
    138Magazine
    140Week in Review
    142Multimedia/Photos
    144Learning Network
  147Services (B)
    151Archive
    153Classifieds
    155College
    157Book a Trip
    159Personals
```

Notes: (a) Part of the domain object model tree of the *New York Times* homepage
(b) its binary partition semantic tree
(c) the converted *Group* tree
(d) the final hierarchical partition tree after promotion.

4.3.2 Grouping

Next, we identify and group a sequence of similar binary partitions under Group nodes. The Group nodes are made up of multiple similar Instance nodes as its children.

The grouping algorithm first initialises the type of the leaf nodes in the binary partition tree as ‘simple’. While traversing the tree in post-order, if it finds two ‘simple’ sibling nodes, and if the cost for grouping these two nodes is less than a fixed threshold 0.33 (according to the cost factor evaluation discussed in the previous section), then it marks these nodes as ‘Instances’ and their parent as a ‘Group’ node. For example, in Figure 7, nodes ‘Sports’ and ‘Health’ are such sibling nodes. Similarly, if it finds two sibling nodes that are marked as ‘Group’, and if the cost for grouping their instances is less than the threshold, then it marks the parent of these sibling nodes as ‘Group’ and merges their ‘Instances’. For example, the parent nodes of ‘Health’ and ‘Sports’, and of ‘Science’ and ‘Technology’, are two such nodes, as seen in Figure 7(c). Alternatively, if one of the sibling nodes is ‘simple’ and the other node is a ‘Group’ node with the above property, then the ‘simple’ node is merged with the ‘Group’ node. Figure 7 shows the conversion of a binary partition tree into a Group Tree. Column 2 and Column 3 represent the binary partition and Group trees, respectively.

4.3.3 Promotion

The final step in hierarchical partitioning is promotion. After grouping, all the contents of the web page are still at the leaf nodes, and hence promotion of some of the leaf nodes is necessary in order to organise them into a hierarchy. The promotion algorithm identifies those leaf nodes that should be promoted above their siblings. A node is considered for promotion if it satisfies one of the following rules:

Insider promotion

An EMPHASISED node can be promoted if it is the first child of its parent and the parent is not marked as a ‘Group’ node. A node is EMPHASISED if it satisfies one of the following conditions:

- There is a bold tag like <b>, <bold> and <h1> on its path from the root node.
- A bold tag is defined for this node’s ‘class’ attribute.
- The node’s labelled text is fully capitalised.
- There is a <ul> or <ol> tag in the path to the next consecutive node.

Outsider promotion

A node can be promoted if it satisfies all of the following conditions:

- It is the first child of its parent.
- Its parent is not marked as ‘Group’ node.
- Its only sibling is either a ‘Group’ node or singleton.

The nodes satisfying the EMPHASISED conditions are marked with (B) in Figure 7c. Upon insider promotion, the EMPHASISED node replaces its parent ‘Partition Node’. If the promotion rule reapplies, the EMPHASISED node is promoted once again. Figure 7c represents the Group Tree and Figure 7d represents the final Hierarchical Partition Tree after promotion. For example, the ‘News’ node in Figure 7c is EMPHASISED, and it is the first child of its parent. So using insider promotion, it is promoted above all its sibling nodes (‘International’ through ‘Corrections’). The nodes ‘Opinion’ and ‘Features’ are also promoted using the same rule.

5 Taxonomy mining

Taxonomy mining involves several tasks, including:

- Separating important concepts (the categories that define the context) from instances (the contents that are members of each concept) and human-oriented decoration, such as navigational panels and advertisement bars.
- Mining taxonomical relationships among the concepts.

Various phases involved in taxonomy mining are explained in the following subsections.

5.1 Frequent label mining

The frequent label miner preprocesses the home pages using a semantic partitioner to generate label hierarchies that are used to mine the taxonomy. Different websites often repeat important labels for a given domain. Our system exploits this to identify important labels as candidate concepts. By using the input support figure, we separate candidate concepts from the rest. For example, in the news domain, our system identifies business, sports, politics, technology, health and entertainment as candidate concepts.

This simple frequency-based mining might miss some important labels that are relevant but infrequent. For example, in http://www.washtimes.com/, our system identified ‘Entertainment’ to be a frequent label but missed ‘Civil war’ and ‘Culture’. To find such relevant labels, our system utilises the semantic partition tree to also collect all group siblings of all frequent labels as candidate concepts. For example, upon querying the group of the ‘Entertainment’ label, our system identifies the missing candidate concepts ‘Civil war’ and ‘Culture’.

5.2 Abridgement phase

Not all candidate concepts identified by the frequent label miner are relevant for the domain – for example, ‘NYT Store’ in http://nytimes.com/. To eliminate such labels, we introduce the following rule. A candidate concept is eliminated if:

- it is not a link label
- its URL refers outside the domain
- the page referred to does not contain any new candidate concepts or valid instances, as defined in Section 6.

The remaining candidate concepts are identified to be relevant domain concepts.

5.3 Mapping labels to concepts

During this phase, similar concept labels are grouped together based on their lexicographic similarity. The words are stemmed using Porter's stemming algorithm (Porter, 1980). Next, Jaccard's similarity coefficient (Korfhage, 1999), calculated as:

$$\frac{|X \cap Y|}{|X \cup Y|}$$

where X and Y are sets of stemmed words from two different labels, is used to group similar labels. We denote each collection of labels to be a *concept*, and the labels are recorded using the *Syn* function. This simple similarity measure groups labels that are lexicographically related (such as 'Sport' and 'Sports'), but not those that are only semantically related (such as 'World' and 'International'). The issue of identifying and grouping semantically related labels remains to be investigated as future work.

5.4 Mining taxonomical relationships

The concepts obtained from the mapping phase are flat. To organise them into a taxonomy, we need to infer hierarchical relationships among them. Using the algorithm outlined in Algorithm 1, we mine these relationships from the semantically partitioned web pages. In this algorithm, *FR* refers to the bag (a collection of objects whose members need not be distinct) of frequent relationships, and *NFR* refers to the bag of nonfrequent relationships. Two concepts a and b in a tree are *i-related* if a is an ancestor of b and i nodes connect them in the tree. We first mine 1-*related* pairs, which are direct parent-child relationships, and find the frequent relationships. Next, we follow the same procedure for the union of infrequent 1-*related* pairs and all 2-*related* pairs to find more frequent is-a relationships. We repeat this procedure until we reach the maximum depth of the input trees available for mining. For example, suppose *a–b* and *a–c–b* are paths from two trees. The maximum depth would be two. If the necessary support for frequent pairs is two, then $a–b \in$ 1-*related*, $FR_1 = \phi$, and $a - b \in| NFR_1$. In the second iteration, we find that $a–b \in|$ 2-*related* and the bag R contain *a–b* twice, so $FR_2 = \{a–b\}$. Given the collections of concepts and the hierarchical partition trees corresponding to the relevant home pages, this algorithm produces the concept taxonomy.

Algorithm 1: Mine taxonomical relationship

TaxMiner
Input: C: Set of concepts, S: Set of semantically partitioned web pages, Sup: Support
Output: Tree representing the hierarchy of concepts

```
1: $R_0 := \phi$; NFR$_0$: = $\phi$
2: $d$:= max(depth($s \in S$))
3: **for** $i$ = 1 to $d$ **do**
4:     $R_i \leftarrow NFR_{i-1} \cup_{bag} i - related\ (S)$
5:     $FR_i$:= frequent ($R_i$)
6:     $NFR_i$: = non-frequent ($R_i$)
7: **end for**
8: Return $U_{i=1}^{d} FR_i$
```

5.5 Expanding the taxonomy beyond home pages

The taxonomy obtained from the previous steps still corresponds to the home pages. To expand the domain taxonomy deeper, we follow the links corresponding to every concept label, and expand the taxonomy by repeating the earlier phases, thus identifying subconcepts. For example, ‘Sports’ is a concept in the taxonomy obtained from the home pages. If we follow all the links corresponding to the ‘Sports’ concept and repeat the above steps, the subtaxonomy for the ‘Sports’ concept, which contains subconcepts such as ‘Baseball’, ‘Tennis’ and ‘Horse racing’, is obtained.

Following the links from all known concepts and mining them yields the final taxonomy for the domain.

6 Instance extraction

A taxonomy schema is made up of a set of hierarchical relationships between concepts. To populate a taxonomy, OntoMiner must identify the concept instances. As mentioned earlier, instances correspond to members of concepts. OntoMiner’s instance extraction algorithm can extract relational instances made up of attribute-value pairs, as well as complex, semi-structured instances from a collection of domain-relevant websites. Whenever they are available, it also attempts to extract the attributes’ labels.

The key components in the instance extraction module are the identification and separation of instances, extraction of attribute information from the instances and labelling of the instances.

6.1 Identification of instances

The instance extraction procedure first identifies the ‘instance template’ from the concept web page and uses this template to extract instances by following all the links of that page. The Semantic Partitioning algorithm, of Section 4, is used to arrange the labels in the concept web page into groups. From each group, the instance extraction procedure fetches any two links and flat-partitions them using the algorithm described in Section 4.1 to obtain two sequences of logical segments. Next, it aligns these two segment sequences using the Levenshtein (1966) distance measure, and Jaccard’s content similarity measure is used to identify similar and dissimilar logical segments. The aligned similar segments correspond to ‘human-oriented decoration’, such as navigational panels and advertisement bars, where aligned but dissimilar segments correspond to instance information. For example, in Figure 1, the segment alignment is $\forall i\ a_i$ aligns with c_i. Since a_3 and c_3 are aligned but dissimilar segments, they denote instance information. The rest of the segments denote headers and navigation bars.

6.2 Extraction of data from instances

The root-to-leaf tag paths of the leaf nodes within the dissimilar logical segments become the set of paths that characterise the group’s instances. Such sets of paths are extracted from every group within the concept web page. The group path set with the most frequently occurring paths among all groups is identified to be the ‘instance template’.

For example, in Figure 1, given the concept web page, it identified the candidate segment as the group that contains the instance template. Next, the instance extraction procedure uses the instance template to extract the concept instances from every group in the concept web page. As shown in Figure 1, it extracts the concept instances from inside the solid coloured boxes from the three instance web pages.

6.3 Labelling the instances

Once the segments corresponding to the content instance is identified, the instance extraction procedure uses the Semantic Partitioning algorithm to arrange the labels within these segments into a hierarchy. During this phase of the Semantic Partitioning, an additional promotion rule is used that allows the promotion of frequent labels, whenever possible. Once the Semantic Partitions are obtained for the instance segments, the promoted labels are recorded as the attribute labels and their children are recorded as their values.

7 Experimental results

In this section, we provide experimental results for Data Table Detector, Data Table Type Identifier, Hierarchical Partitioning Algorithm, Taxonomy Mining Algorithm and Instance Extraction Algorithm. The test bed for the experiments is collected from news, biology, travel and shopping domains. First, we present metrics for evaluating the performance of HP and taxonomy mining algorithms and then present individual results for each component.

7.1 Evaluation

We use the precision and recall metrics for performance evaluations. The precision and recall of Hierarchical Partitioning (HP) and taxonomy mining algorithms can be calculated by comparing the transitive closure of parent-child relationships inferred by the ‘algorithmically generated’ hierarchies with those implied by the ‘gold-standard’ hierarchies. The ‘gold-standard’ hierarchy for each page and taxonomy was created manually, and then the transitive closure of all parent-child relationships was calculated. The precision and recall are calculated by the following:

$$Precision = \frac{\{R\} \bigcap \{R'\}}{\{R\}}, \quad Recall = \frac{\{R\} \bigcap \{R'\}}{\{R'\}}$$

where *R* and *R’* are the sets of transitive closure of parent-child relationships implied by the ‘algorithmically generated’ and ‘gold-standard’ hierarchies, respectively.

7.2 Flat Partitioner (FP)

The precision for the Page Segmentor is calculated by comparing the number of correct segments identified by the algorithm to the total number of segments identified. Similarly, the recall is calculated as the ratio of the number of correct segments identified to the total number of actual segments. The correct segments in the web page were identified by an evaluator:

$$Precision = \frac{S_{ci}}{S_{ci} + S_{wi}}$$

$$Recall = \frac{S_{ci}}{S_{ci} + S_{uc}}$$

where:

S_{ci} – is the correctly identified segments

S_{wi} – is the wrongly identified segments

S_{uc} – is the unidentified correct segments.

The segments are nothing but boundary lines in the DOM tree of the web page. A window of five units on both sides of the boundary lines is allowed for errors in calculation. Table 5 shows the experimental results for the Page Segmentor. The relatively low precision suggests that the page segmentation component found more segments than the ideal list of segments in those pages. The objective of the semantic partitioner is not to find ideal logical segments but to find ideal relationships among the contents of these web pages. Hence, even with a relatively low precision of 60% for the news category, the precision for hierarchical partitioner on news category is high at 77%.

Table 5 Precision and recall for the page segmentation algorithm

Category	*Domain*	S_c	S_{ci}	S_{wi}	S_{ui}	*Precision (%)*	*Recall (%)*
News	www.abcnews.go.com	12	12	17	0	71	100
	www.bbc.co.uk	15	13	9	2	60	87
	www.cnn.com	19	14	10	5	59	74
	news.google.com	10	9	6	1	60	90
	www.reuters.com	14	11	8	3	58	79
	www.latimes.com	18	16	9	2	64	89
	newsbot.msnbc.msn.com	12	11	8	1	58	92
	www.nytimes.com	13	11	7	2	61	85
	www.usatoday.com	15	12	22	3	55	80
	www.washingtonpost.com	29	21	19	8	53	72
	Average					60	85
Overall average						71	88

7.3 Data Table Detector (DTD)

The precision and recall of the data table detection component is obtained by checking whether the identified data tables are actually data tables. The precision is calculated as the ratio of the number of data tables that were identified correctly to the total number of data tables that were identified by the data table detection component. Similarly, the recall is calculated as the ratio of the number of data tables correctly identified by the total number of data tables present in the web page:

$$Precision = \frac{DT_{ci}}{DT_{ci} + DT_{wi}}$$

$$Recall = \frac{DT_{ci}}{DT_{ci} + DT_{uc}}$$

where:

DT_{ci} – is the number of data tables that were identified correctly in a web page

DT_{wi} – is the number of data tables that were identified but were actually wrong in the web page

DT_{uc} – is the unidentified correct data tables.

The precision and recall values for data table detection are presented in Table 6. The precision of the DTD for certain web pages is low, owing to the inability of the HTML Tidy[3] parser to parse those web pages. Also, the heuristic rules do not apply to certain web pages that present nonrelational data in tabular format.

Table 6 Experimental results for DTD

Website address	DT_c	DT_{ci}	DT_{wi}	DT_{uc}	*Precision (%)*	*Recall (%)*
www.cem.msu.edu/reusch/…/sovent.htm	2	2	1	0	67	100
www.cem.msu.edu/reusch/…/equienr.htm	1	0	1	0	0	0
www.cem.msu.edu/reusch/…/confengy.htm	1	1	0	0	100	100
wp.netscape.com/…/table_sample.html	9	9	0	0	100	100
wulfenite.fandm.edu/…/Table_20.html	3	3	0	0	100	100
uaweb.arizona.edu/…/data-tables.shtml	4	4	1	0	80	100
www.geocities.com/tablizer/cntrl1.htm	5	2	3	3	40	40
www.josseybass.com/…/tables.htm	4	4	0	0	100	100
www.oesr.qld.gov.au/…/table1290.htm	1	1	0	0	100	100
www.swenergy.org/…/tucson_topsc.htm	2	0	2	2	0	0
rugsusa.com/braidedrugs.html	1	1	0	0	100	100
www.cnn.com	9	7	2	2	78	78
Java.sun.com/j2se/1.4.2/…/Byte.html	3	3	0	0	100	100
www.asu.edu/registrar/…/finals.html	6	6	1	0	86	100
Average					75	80

7.4 Data Table Type Identifier (DTTI)

The Data Table Type Identifier is evaluated based on the number of data tables that were identified correctly. The precision and recall of the DTTI are the same since the DTTI identifies some orientation for every table with which it has been provided. The precision/recall for each type of orientation is presented individually. The precision/recall for a type of orientation is calculated as the ratio of the number of orientation of this type identified correctly to the total number of tables of the corresponding type:

$$Precision / Recall = \frac{O_{ci}}{N}$$

where O_{ci} is the number of orientation (of the current type) identified correctly and N is the number of tables of the corresponding type. The precision/recall for DTTI is presented in Table 7. The overall precision/recall is low because the extracted features from each cell are not sufficient to separate the header from the data and, hence, a row-column-oriented table is recognised as column oriented. Also, the clustering algorithms did not always cluster the header cells together. These values can be improved by extracting more features and using better clustering algorithms in future.

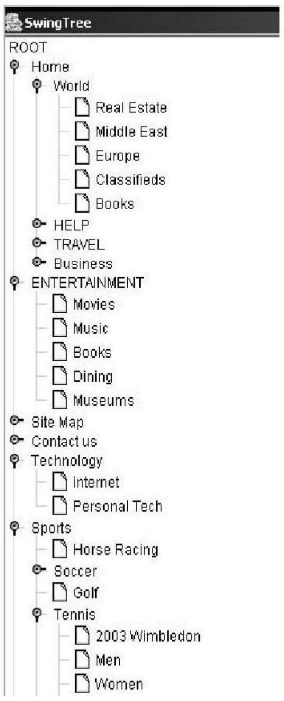
Table 7 Experimental results for DTTI

Table type	*N*	O_{ci}	*Precision/recall (%)*
Column-oriented	28	22	78
Row-oriented	2	1	50
Row-column-oriented	4	3	75
Multiple header column-oriented	1	1	100
Multiple header row-oriented	1	1	100
Multiple header row-column-oriented	2	1	50
No-orientation	5	4	80
Average			79

7.5 HP algorithm

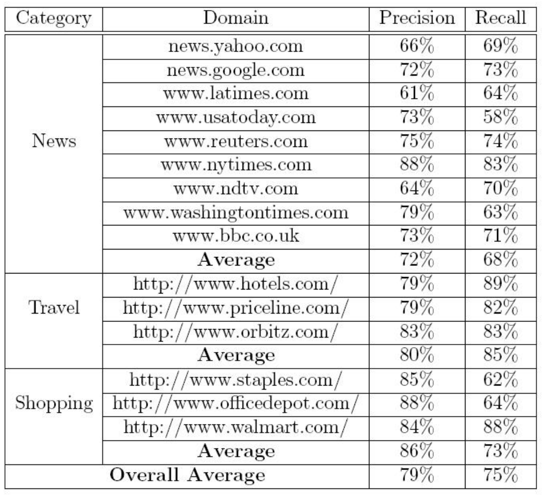
The experimental results for the semantic partitioning are as shown in the table to the right in Figure 8. For news, HP performs with an overall precision of 87% and a recall of 82%. HP on the travel category performs with overall 87% precision and 94% recall. The experiments for the shopping perform with overall 96% precision and only 73% recall. The recall of web pages in the shopping category can be improved by tuning the promotion rules and by using a better data table detector.

Figure 8 A fragment of the taxonomy obtained for the news domain and the results for the HP algorithm



SwingTree

- ROOT
  - Home
    - World
      - Real Estate
      - Middle East
      - Europe
      - Classifieds
      - Books
    - HELP
    - TRAVEL
    - Business
  - ENTERTAINMENT
    - Movies
    - Music
    - Books
    - Dining
    - Museums
  - Site Map
  - Contact us
  - Technology
    - Internet
    - Personal Tech
  - Sports
    - Horse Racing
    - Soccer
    - Golf
    - Tennis
      - 2003 Wimbledon
      - Men
      - Women



Category	Domain	Precision	Recall
News	news.yahoo.com	66%	69%
	news.google.com	72%	73%
	www.latimes.com	61%	64%
	www.usatoday.com	73%	58%
	www.reuters.com	75%	74%
	www.nytimes.com	88%	83%
	www.ndtv.com	64%	70%
	www.washingtontimes.com	79%	63%
	www.bbc.co.uk	73%	71%
	Average	72%	68%
Travel	http://www.hotels.com/	79%	89%
	http://www.priceline.com/	79%	82%
	http://www.orbitz.com/	83%	83%
	Average	80%	85%
Shopping	http://www.staples.com/	85%	62%
	http://www.officedepot.com/	88%	64%
	http://www.walmart.com/	84%	88%
	Average	86%	73%
Overall Average		79%	75%

Table 1: Precision and Recall for the Semantic Partitioning algorithm.

7.6 Taxonomy mining algorithm

Figure 8 shows a fragment of the mined taxonomy for the news domain. The precision and recall values for the generated taxonomy are 91% and 79%, respectively. The high precision indicates that the taxonomy mining algorithm is able to identify the relevant concepts and their labels from the websites accurately. The relatively lower recall value shows that the algorithm misses some of the relevant domain concepts. This is caused by the grouping errors due to irregularities in the presentation of the concepts within web pages.

7.7 Instance extraction algorithm

Table 8 presents the precision and recall values for instance extraction from our experiments.

Table 8 Precision and recall values for instances extraction

Task	*Precision (%)*	*Recall (%)*
Extracting unlabelled attribute value for news instances	81	97
Extracting instances from concept pages of travel pages	85	84
Identifying the correct attribute value pairs from the travel pages	80	91

7.8 Discussion

It can be seen that the recall and precision values are reasonably good for the travel websites, since these websites present similar information in a regular fashion. The high recall in all the cases indicates that the instance extraction algorithm is able to extract the attribute labels and data values. The lower precision in some cases is due to nested data tables, which are currently not handled by the data table detector.

8 Conclusions and future work

We present algorithms to logically partition a web page and infer the hierarchy among the labels in the web page, mine the taxonomy of important concepts from overlapping domain specific websites, and extract attributed instances and their values corresponding to each concept in the taxonomy.

The novel cost factors used in the semantic partitioning algorithm help to infer the logical hierarchical organisation of the contents within a single web page. OntoMiner requires only a collection of domain-specific sites in building the taxonomy of concepts, as it uses frequency-based techniques to mine the is-a relationships. OntoMiner can find complex semistructured instances for attributed concepts and can extract labels for attributes whenever they are present. Overall, OntoMiner attains a precision of 90% and a recall of more than 80% for semantic partitioning, taxonomies, attributes and values of their instances. Following are some of the future directions of our work. The five cost factors do not always perform well for all kinds of web pages. We plan to tune the cost factors in order to achieve better overall hierarchical partitioning. We also plan to investigate techniques that combine syntactic as well as semantic regularities (Mukherjee *et al.*, 2003), and we will try to iteratively repair hierarchical partitions with bootstrapped ontologies to yield higher precision and recall. Currently, we mine for the is-a relationships among the concepts from each of the semantic trees and build the taxonomy of concepts. Instead, we could use ontology mapping and merging techniques to obtain a generic representative taxonomy for the domain based on the input web pages.

References

Arasu, A. and Garcia-Molina, H. (2003) 'Extracting structured data from web pages', *ACM SIGMOD Conference on Management of Data*, San Diego, USA.

Arocena, G.O. and Mendelzon, A.O. (1998) 'Weboql: restructuring documents, databases, and webs', *International Conference on Data Engineering*.

Bertino, E., Guerrini, G., Merlo, I. and Mesiti, M. (1999) 'An approach to classify semi-structured objects', *Proceedings of the European Conference on Object-Oriented Programming*, LNCS 1628.

Chen, H-H., Tsai, S-C. and Tsai, J-H. (2002) 'Mining tables from large scale html texts', *18th International Conference on Computational Linguistics*, Saabrucken, Germany.

Chung, C.Y., Gertz, M. and Sundaresan, N. (2002) 'Reverse engineering for web data: from visual to semantic structures', *International Conference on Data Engineering*.

Cong, G., Yi, L., Liu, B. and Wang, K. (2002) 'Discovering frequent substructures from hierarchical semi-structured data', *Proceedings of the Second SIAM International Conference on Data Mining*.

Craven, M., DiPasquo, D., Freitag, D., McCallum, A.K., Mitchell, T.M., Nigam, K. and Slattery, S. (1998) 'Learning to extract symbolic knowledge from the World Wide Web', *National Conference on Artificial Intelligence*, Menlo Park, Madison, US: AAAI Press, pp.509–516.

Crescenzi, V., Mecca, G. and Merialdo, P. (2001) 'Roadrunner: towards automatic data extraction from large web sites', *Proceedings of 27th International Conference on Very Large Data Bases*, pp.109–118.

Dill, S., Eiron, N., Gibson, D., Gruhl, D., Guha, R., Jhingran, A., Kanungo, T., Rajagopalan, S., Tomkins, A., Tomlin, J.A. and Zien, J.Y. (2003) 'Semtag and seeker: bootstrapping the semantic web via automated semantic annotation', *International Conference on the World Wide Web*.

Doan, A., Domingos, P. and Halevy, A. (2003) 'Learning to match the schemas of data sources: a multistrategy approach', *Machine Learning*.

Doorenbos, R.B., Etzioni, O. and Weld, D.S. (1997) 'A scalable comparison-shopping agent for the World Wide Web', in W. Lewis Johnson and B. Hayes-Roth (Eds.) *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, Marina del Rey, CA, USA: ACM Press, pp.39–48.

Embley, D.W., Campbell, D.M., Smith, R.D. and Liddle, S.W. (1998) 'Ontology-based extraction and structuring of information from data-rich unstructured documents', *International Conference on Knowledge Management*.

Etzioni, O., Cafarella, M., Downey, D., Kok, S., Popescu, A-M., Shaked, T., Soderland, S., Weld, D.S. and Yates, A. (2004) 'Web-scale information extraction in knowitall (preliminary results)', *International Conference on the World Wide Web.*

Garofalakis, M., Gionis, A., Rastogi, R., Seshadri, S. and Shim, K. (2000) 'Xtract: a system for extracting document type descriptors from xml documents', *ACM SIGMOD Conference on Management of Data.*

Hammer, J., Garcia-Molina, H., Nestorov, S., Yerneni, R., Breunig, M.M. and Vassalos, V. (1997) 'Template-based wrappers in the tsimmis system', *ACM SIGMOD Conference on Management of Data.*

Hong, T.W. and Clark, K.L. (2001) 'Using grammatical inference to automate information extraction from the web', *Principles of Knowledge Discovery and Data Mining.*

Hurst, M. (2002) 'Classifying table elements in html', *11th International World Wide Web Conference*.

Korfhage, R.R. (1999) *Information Storage and Retrieval*, New York: JohnWiley Computer Publications.

Kushmerick, N., Weld, D.S. and Doorenbos, R.B. (1997) 'Wrapper induction for information extraction', *International Joint Conference on Artificial Intelligence (IJCAI)*, pp.729–737.

Lerman, K., Getoor, L., Minton, S. and Knoblock, C. (2004) 'Using the structure of web sites for automatic segmentation of tables', *ACM SIGMOD Conference on Management of Data.*

Levenshtein, V. (1966) 'Binary codes capable of correcting deletions, insertions and reversals', *Soviet Physics Daklady*, Vol. 10, pp.707–710.

Madhavan, J., Bernstein, P., Halevy, A. and Domingos, P. (2002) 'Representing and reasoning about mappings between domain models', *Proceedings of AAAI-02, 18th Conference of the American Association for Artificial Intelligence*, Menlo Park, US: AAAI Press.

Maedche, A. and Staab, S. (2000) *The Text-to-Onto Ontology Learning Environment.*

McBride, B. (2002) 'Four steps towards the widespread adoption of a semantic web', *International Semantic Web Conference*.

McDowell, L., Etzioni, O., Halevy, A., Levy, H., Gribble, S., Pentney, W., Verma, D. and Vlasseva, S. (2003) 'Mangrove: enticing ordinary people onto the semantic web via instant gratification', *International Semantic Web Conference*.

Mukherjee, S., Yang, G. and Ramakrishnan, I.V. (2003) 'Annotating content-rich web documents: structural and semantic analysis', *International Semantic Web Conference*.

Muslea, I., Minton, S. and Knoblock, C. (1999) 'A hierarchical approach to wrapper induction', *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pp.190–197.

Navigli, R., Velardi, P. and Gangemi, A. (2003) 'Ontology learning and its application to automated terminology translation', *IEEE Intelligent Systems*, January, Vol. 18, No. 1.

Nestorov, S., Abiteboul, S. and Motwani, R. (1998) 'Extracting schema from semistructured data', *ACM SIGMOD Conference on Management of Data*.

Noy, N. and Musen, M. (2000) 'Prompt: algorithm and tool for automated ontology merging and alignment', *Proceedings of AAAI-2000, 17th Conference of the American Association for Artificial Intelligence*, Menlo Park, US: AAAI Press.

Papakonstantinou, Y. and Vianu, V. (2000) 'Dtd inference for views of xml data', *ACM Symposium on Principles of Database Systems*.

Porter, M.F. (1980) 'An algorithm for suffix stripping', *Program*, Vol. 14, No. 3, pp.130–137.

Riloff, E. and Shepherd, J. (1997) 'A corpus-based approach for building semantic lexicons', in C. Cardie and R. Weischedel (Eds.) *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, Somerset, New Jersey, pp.117–124.

Theobald, M., Schenkel, R. and Weikum, G. (2003) 'Exploiting structure, annotation, and ontological knowledge for automatic classification of xml data', *ACM SIGMOD Workshop on the Web and Databases (WebDB)*.

Wang, Y. and Hu, J. (2002) 'A machine learning based approach for table detection on the web', *11th World Wide Web Conference*.

Yang, J., Choi, J. and Oh, H. (2001) *Morpheus: A Customized Comparison-shopping Agent*.

Yang, Y. and Zhang, H. (2001) 'Html page analysis based on visual cues', *International Conference on Document Analysis and Recognition*.

Zaki, M.J. (2002) 'Efficiently mining frequent trees in a forest', *8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, July.

Notes

1 www.yahoo.com

2 www.dmoz.org

3 Jtidy parser, http://www.sourceforge.net/projects/jtidy