

Mining Data Records in Web Pages

Bing Liu

Department of Computer Science
University of Illinois at Chicago
851 S. Morgan Street
Chicago, IL 60607-7053

liub@cs.uic.edu

Robert Grossman

Dept. of Mathematics, Statistics, and
Computer Science
University of Illinois at Chicago
851 S. Morgan Street, IL 60607

grossman@uic.edu

Yanhong Zhai

Department of Computer Science
University of Illinois at Chicago
851 S. Morgan Street
Chicago, IL 60607-7053

yzhai@cs.uic.edu

ABSTRACT

A large amount of information on the Web is contained in regularly structured objects, which we call *data records*. Such data records are important because they often present the essential information of their host pages, e.g., lists of products or services. It is useful to mine such data records in order to extract information from them to provide value-added services. Existing automatic techniques are not satisfactory because of their poor accuracies. In this paper, we propose a more effective technique to perform the task. The technique is based on two observations about data records on the Web and a string matching algorithm. The proposed technique is able to mine both contiguous and non-contiguous data records. Our experimental results show that the proposed technique outperforms existing techniques substantially.

Categories and Subject Descriptors

I.5 [Pattern Recognition]: statistical and structural
H.2.8 [Database Applications]: data mining

Keywords

Web data records, Web mining, Web information integration

1. INTRODUCTION


A large amount of information on the Web is presented in regularly structured objects. A list of such objects in a Web page often describes a list of similar items, e.g., a list of products or services. In this paper, we call them *data records*. Mining data records is useful because it allows us to integrate information from multiple sources to provide value-added services. Figure 1 gives an example, which is a segment of a Web page that lists two Apple notebooks. The full description of each notebook is a data record. The objective of this work is to automatically mine all the data records in a given Web page.

Several semi-automatic and automatic approaches have been reported in the literature for mining data records (or their boundaries) from Web pages, e.g., [2][3][4][6][8][9][10][12]. [4][8][9][12] use the machine learning approach, which is semi-automatic as it requires human labeling of specific regions in the Web page to mark them as interesting. [6] presents an automatic method that uses a set of heuristics and domain ontology to perform the task. [2] extends this approach by designing some

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGKDD '03, August 24-27, 2003, Washington, DC, USA
Copyright 2003 ACM 1-58113-737-0/03/0008...\$5.00.

additional heuristics without using any domain knowledge. We will show in the experiment section that the accuracy of this approach is poor. [3] proposes another automatic method, which uses Patricia tree [7] and approximate sequence alignment to find patterns (which represent a set of data records) in a Web page. Due to the inherent limitation of Patricia tree and inexact sequence matching, it often produces many patterns and most of them are spurious. Again, this method performs poorly. [10] proposes a clustering and grammar induction based method. However, the results are not very satisfactory [10].

1.  [Apple iBook Notebook M8600LL/A \(600-MHz PowerPC G3, 128 MB RAM, 20 GB hard drive\)](#)
Buy new: \$1,194.00
Usually ships in 1 to 2 days
Customer Rating: ★★★★★
Best use: (what's this?) Business: ●●●● Portability: ●●●● Desktop Replacement: ●●●● Entertainment: ●●●●
600 MHz PowerPC G3, 128 MB SRAM, 20 GB Hard Disk, 24x CD-ROM, AirPort ready, and Mac OS X, Mac OS X, Mac OS 9.2, Quick Time, iPhoto, iTunes 2, iMovie 2, AppleWorks, Microsoft IE


2.  [Apple Powerbook Notebook M8591LL/A \(667-MHz PowerPC G4, 256 MB RAM, 30 GB hard drive\)](#)
Buy new: \$2,399.99
Customer Rating: ★★★★★
Best use: (what's this?) Portability: ●●●● Desktop Replacement: ●●●● Entertainment: ●●●●
667 MHz PowerPC G4, 256 MB SDRAM, 30 GB Ultra ATA Hard Disk, 24x (read), 8x (write) CD-RW, 8x; included via combo drive DVD-ROM, and Mac OS X, QuickTime, iMovie 2, iTunes(6), Microsoft Internet Explorer, Microsoft Outlook Express, ...

Figure 1. An example: two data records

Apart from low accuracy, existing approaches also assume that relevant information of a data record is contained in a contiguous segment of the HTML code. This model is insufficient because in some Web pages, the description of one object (a data record) may intertwine with the descriptions of some other objects. For example, the descriptions of two objects in the HTML source may follow this sequence, part1 of object1, part1 of object2, part2 of object1, part2 of object2. Thus, the descriptions of both object1 and object2 are not contiguous. However, when they are displayed on a Web browser, they appear contiguous to human viewers.

In this paper, we propose a novel and more effective method to mine data records in a Web page automatically. The algorithm is called MDR (Mining Data Records in Web pages). It currently finds all data records formed by table and form related tags, i.e., table, form, tr, td, etc. A large majority of Web data records are formed by them. Our method is based on two observations:

1. A group of data records that contains descriptions of a set of similar objects are typically presented in a particular region of a page and are formatted using similar HTML tags. Such a region is called a *data region*. For example, in Figure 1 two notebooks are given in one region. They are also formatted

using almost the same sequence of HTML tags. If we regard the HTML tags of a page as a string, we can use string matching to compare different sub-strings to find those similar ones, which may represent similar objects/data records.

The problem with this approach is that the computation is prohibitive because a data record can start from anywhere and end anywhere. A set of data records typically do not have the same length in terms of their tag strings because they may not contain exactly the same pieces of information (see Figure 1).

- The nested structure of HTML tags in a Web page naturally forms a *tag tree*. Our second observation is that a group of similar data records being placed in a specific region is reflected in the tag tree by the fact that they are under one parent node, although we do not know which parent (our algorithm will find out). For example, the tag tree for the page in Figure 1 is given in Figure 2 (some details are omitted). Each notebook (a data record) in Figure 1 is wrapped in 5 TR nodes with their sub-trees under the same parent node TBODY (Figure 2). The two data records are in the two dashed-lined boxes. In other words, a set of similar data records are formed by some child sub-trees of the same parent node.

A further note is that it is very unlikely that a data record starts inside of a child sub-tree and ends inside another child sub-tree. Instead, it starts from the beginning of a child sub-tree and ends at the same or a later child sub-tree. For example, it is unlikely that a data record starts from TD* and ends at TD# (Figure 2). This observation makes it possible to design a very efficient algorithm to mine data records.

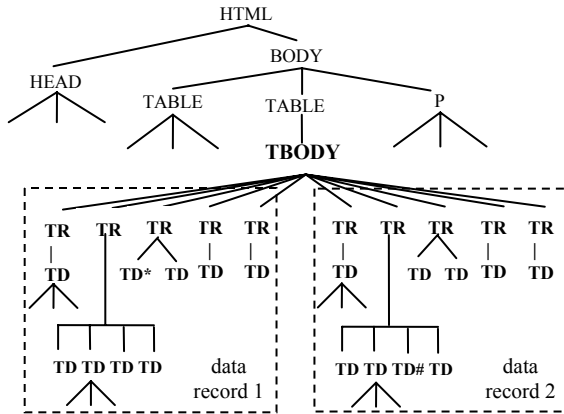


Figure 2. Tag tree of the page in Figure 1

Our experiments show that these observations are true. Note that we do not assume that a Web page has only one data region with data records. In fact, a Web page may contain a few data regions. Different regions may have different data records. Our method only requires that a data region to have two or more data records.

Given a Web page, the proposed technique works in three steps:

- Step 1: Building a HTML tag tree of the page.
- Step 2: Mining data regions in the page using the tag tree and string comparison. Note that instead of mining data records directly, which is hard, our method mines data regions first and then find the data records within them. For example, in Figure 2, we first find the single data region below node TBODY.
- Step 3: Identifying data records from each data region. For example, in Figure 2, this step finds data record 1 and data record 2 in the data region below node TBODY.

2. THE PROPOSED TECHNIQUE

We now present the three steps in the proposed method in turn.

2.1 Building the HTML Tag Tree

In this work, we only use tags in string comparison to find data records. Most HTML tags work in pairs. Each pair consists of an *opening* tag and a *closing* tag. Within each corresponding tag-pair, there can be other pairs of tags, resulting in nested blocks of HTML codes. Building a *tag tree* from a Web page using its HTML code is thus natural. In our tag tree, each pair of tags is considered as one *node*. An example tag tree is shown in Figure 2.

2.2 Mining Data Regions

This step mines every data region in a Web page that contains similar data records. Instead of mining data records directly, which is hard, we first mine *generalized nodes* in a page. A sequence of adjacent generalized nodes forms a data region. From each data region, we will identify the actual data records.

Definition: A *generalized node* (or a *node combination*) of length r consists of r ($r \geq 1$) nodes in the HTML tag tree with the following two properties:

- 1) the nodes all have the same parent.
- 2) the nodes are adjacent.

The reason that we introduce the generalized node is to capture the situation that an object (or a data record) may be contained in a few sibling tag nodes rather than one. For example, in Figures 1 and 2, we can see that each notebook is contained in five table rows (or 5 TR nodes). Note that we call each node in the HTML tag tree a *tag node* to distinguish it from a generalized node.

Definition: A *data region* is a collection of two or more generalized nodes with the following properties:

- 1) the generalized nodes all have the same parent.
- 2) the generalized nodes all have the same length.
- 3) the generalized nodes are all adjacent.
- 4) the normalized edit distance (string comparison) between adjacent generalized nodes is less than a fixed threshold.

For example, in Figure 2, we can form two generalized nodes, the first one consists of the first 5 children TR nodes of TBODY, and the second one consists of the next 5 children TR nodes of TBODY. It is important to notice that although the generalized nodes in a data region have the same length (the same number of children nodes of a parent node in the tag tree), their lower level nodes in their sub-trees can be quite different. Thus, they can capture a wide variety of regularly structured objects.

To further explain different kinds of generalized nodes and data regions, we make use of an artificial tag tree in Figure 3. For notational convenience, we do not use actual HTML tag names but ID numbers to denote tag nodes in a tag tree. The shaded areas are generalized nodes. Nodes 5 and 6 are generalized nodes of length 1 and they together define the data region labeled 1 if the edit distance condition 4) is satisfied. Nodes 8, 9 and 10 are also generalized nodes of length 1 and they together define the data region labeled 2 if the edit distance condition 4) is satisfied. The pairs of nodes (14, 15) and (16, 17) are generalized nodes of length 2. They together define the data region labeled 3 if the edit distance condition 4) is satisfied.

We end this part with two important notes:

1. In practice, the above definitions are very robust as our experiments show. The key assumption here is that nodes forming a data region are from the same parent, which is reasonable. For example, it is unlikely that a data region starts

- at node 7 and ends at node 14 (see also Figure 2).
- A generalized node may not represent a final data record (see Section 2.3). It will be used to find the final data records.

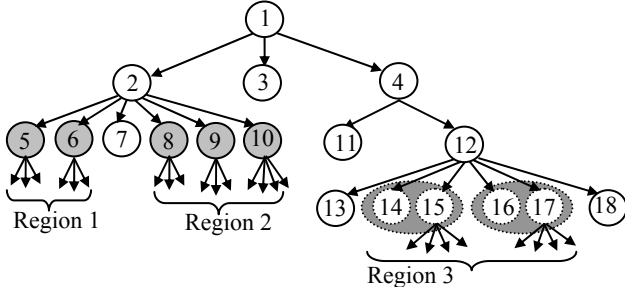


Figure 3: An illustration of generalized nodes and data regions

2.2.1 Comparing Generalized Nodes

In order to find each data region in a Web page, the mining algorithm needs to find the following. (1) Where does the first generalized node of a data region start? For example, in Region 2 of Figure 3, it starts at node 8. (2) How many tag nodes or components does a generalized node in each data region have? For example, in Region 2 of Figure 3, each generalized node has one tag node (or one component).

Let the maximum number of tag nodes that a generalized node can have be K . To answer (1), we can try to find a data region starting from each node sequentially. To answer (2), we can try: one node, two node combination, ..., K node combination. That is, we start from each node and perform all 1-node string comparisons, all 2-node string comparisons, and so on. We then use the comparison results to identify each data region.

The number of comparisons is actually not very large because:

- Due to our assumption, we only perform comparisons among the children nodes of a parent node.
- Some comparisons done for earlier nodes are the same as for later nodes (see the example below).

We use Figure 4 to illustrate the comparison process. Figure 4 has 10 nodes, which are below a parent node, p . We start from each node and perform string comparison of all possible combinations of component nodes. Let the maximum number of components that a generalized node can have be 3 in this example.

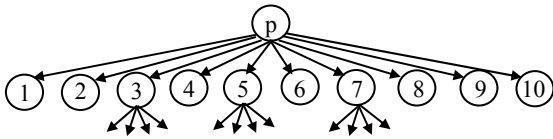


Figure 4: combination and comparison

Start from node 1: We compute the following string comparisons.

- (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 7), (7, 8), (8, 9), (9, 10)
- (1-2, 3-4), (3-4, 5-6), (5-6, 7-8), (7-8, 9-10)
- (1-2-3, 4-5-6), (4-5-6, 7-8-9)

(1, 2) means that the tag string of node 1 is compared with that of node 2. The tag string of a node includes all the tags of its sub-tree, e.g., in Figure 2, the tag string for the second TR node below TBODY is $\langle \text{TR TD TD} \dots \text{TD TD} \rangle$, where “...” denotes the sub-string of sub-tree below the second TD node. The tag string for the third TR node below TBODY is $\langle \text{TR TD TD} \rangle$.

(1-2, 3-4) means that the combined tag string of nodes 1 and 2 is compared with the combined tag string of nodes 3 and 4.

Start from node 2: We only compute:

- (2-3, 4-5), (4-5, 6-7), (6-7, 8-9)
- (2-3-4, 5-6-7), (5-6-7, 8-9-10)

We do not need to do 1-node comparisons because they have been done when we started from node 1 above.

Start from node 3: We only need to compute:

- (3-4-5, 6-7-8)

Here, we do not need to do 1-node and 2-node comparisons because they have been done when we started from node 1.

The overall algorithm (MDR) for computing all the comparisons at each node of a tag tree is given in Figure 5. It traverses the tag tree from the root downward in a depth-first fashion (lines 3 and 4). At each internal node, procedure CombComp (Figure 6) performs string comparisons of various combinations of the children sub-trees. Line 1 says that the algorithm will not search for data regions if the depth of the sub-tree from *Node* is 2 or 1 as it is unlikely that a data region is formed with only a single level of tag(s) (data regions are formed by the children of *Node*).

Algorithm MDR(*Node*, K)

```

1  if TreeDepth(Node) >= 3 then
2    CombComp(Node.Children, K);
3    for each ChildNode ∈ Node.Children
4      MDR(ChildNode, K);

```

Figure 5: The overall algorithm

The main idea of CombComp has been discussed above. Line 3 checks whether there is at least one pair of combinations. If not, no comparison is needed. Lines 4-8 perform string comparisons of various combinations using the edit distance function EditDist.

CombComp(*NodeList*, K)

```

1  for (i = 1; i <= K; i++) /* start from each node */
2    for (j = i; j <= K; j++) /* comparing different combinations
3      if NodeList[i+2*j-1] exists then
4        St = i;
5        for (k = i+j; k < Size(NodeList); k+j)
6          if NodeList[k+j-1] exists then
7            EditDist(NodeList[St..(k-1)], NodeList[k..(k+j-1)]);
8            St = k+j;

```

Figure 6: The structure comparison algorithm

Let N be the total number of nodes in the tag tree, the complexity of MDR is $O(NK)$ without considering string comparison (see [11] for the detailed analysis).

2.2.2 String Comparison Using Edit Distance

The string comparison method that we use is based on the normalized *edit distance* [1][7]. Let the two strings be s_1 and s_2 . The time-complexity of the algorithm is $O(|s_1||s_2|)$ [1]. In our application, the computation can be substantially reduced as we are only interested in very similar strings. The computation is only large when the strings are long. If we want the strings to have the similarity of more than 50%, we can use the following method to reduce the computation: If $|s_1| > 2|s_2|$ or $|s_2| > 2|s_1|$, no comparison is needed because they are obviously too dissimilar.

2.2.3 Determining Data Regions

We now identify each data region by finding its generalized nodes. We use Figure 7 to illustrate the issues. There are 8 data records (1-8) in this page. Our algorithm reports each row as a generalized node and the dash-lined box as a data region.

The algorithm basically uses the string comparison results at each

parent node to find similar children node combinations to obtain candidate generalized nodes and data regions of the parent node. Three main issues are important for making the final decisions.

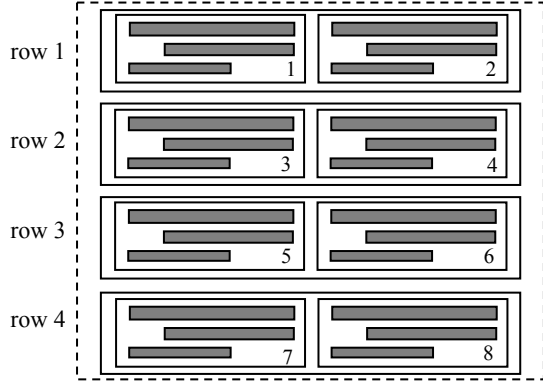


Figure 7. A possible configuration of data records

1. If a higher level data region covers a lower level data region, we report the higher level data region and its generalized nodes. *Cover* here means that a lower level data region is within a higher level data region. For example, in Figure 7, at a lower level we find that cells 1 and 2 are candidate generalized nodes and they together form a candidate data region, row 1. However, they are covered by the data region including all the 4 rows at a higher level. In this case, we only report each row is a generalized node.
2. A property about similar strings is that if a set of strings $s_1, s_2, s_3, \dots, s_n$ are similar to one another, then a combination of any number of them is also similar to another combination of them of the same number. Thus, we only report generalized nodes of the smallest length that cover a data region. In Figure 7, we only report each row as a generalized node rather than a combination of two rows (rows 1-2, and rows 3-4).
3. An edit distance threshold is needed to decide whether two strings are similar. A set of training pages is used to decide it.

The algorithm for this step is given in Figure 8. It finds every data region and its generalized nodes in a page. T is the edit distance threshold. $Node$ is any node. K is the maximum number of tag nodes in a generalized node. $Node.DRs$ is the set of data regions under $Node$, and $tempDRs$ is a temporal variable storing the data regions passed up from every $Child$ of $Node$. Line 1 is the same as line 1 in Figure 5. The idea of the algorithm is to traverse the tag tree top-down in a depth-first fashion. It performs one function at each node when it goes down (line 2), and performs another when it backs up before going down to another tree branch (line 6).

1. When it goes down, at each node it identifies all the candidate data regions of the node using procedure `IdentDRs` (line 2).
2. When it backs up, it checks to see whether the parent level data regions in $Node.DRs$ cover the child level data regions. Those covered child level data regions are discarded. We take the parent level data regions as we believe they are more likely to be true data regions. Those uncovered data regions in $Child.DRs$ are returned and stored in $tempDRs$ (line 6). After all the children nodes of $Node$ are processed, $Node.DRs \cup tempDRs$ gives the current data regions discovered from the sub-tree starting from $Node$ (line 7).

We now discuss procedure `IdentDRs`. Note that the previous step has computed the distance values of all possible child node combinations. This procedure uses these values and the threshold

T to find data regions of $Node$. That is, it needs to decide which combinations represent generalized nodes, where the beginning is and where the end is for each data region.

Algorithm `FindDRs(Node, K, T)`

```

1  if TreeDepth(Node) => 3 then
2  Node.DRs = IdentDRs(1, Node, K, T);
3  tempDRs = ∅;
4  for each Child ∈ Node.Children do
5  FindDRs(Child, K, T);
6  tempDRs = tempDRs ∪ UnCoveredDRs(Node, Child);
7  Node.DRs = Node.DRs ∪ tempDRs

```

Figure 8: Finding all data regions in the tag tree

Procedure `IdentDRs` is given in Figure 9, which ensures the smallest generalized nodes are identified. The `IdentDRs` procedure is recursive (line 15). In each recursion, it extracts the next data region $maxDR$ that covers the maximum number of children nodes. $maxDR$ is described by three members (line 1), (1) the number of nodes in a combination, (2) the location of the start child node of the data region, and (3) the number of nodes involved in or covered by the data region. $curDR$ is the current candidate data region being considered. String comparison results are stored in a data structure attached with each node. The value can be obtained by calling procedure `Distance(Node, i, j)` (which is just a table lookup, and is not listed here), where i represents i -combination, and j represents the j th child of $Node$. `IdentDRs` basically checks each combination (line 2) and each starting point (line 3). For each possibility, it finds the first continuous region with a set of generalized nodes (lines 5-10). Lines 11-12 update the maximum data region $maxDR$. The conditions in line 11 ensure that smaller generalized nodes are used unless the larger ones cover more nodes and starts no later than the smaller ones.

Procedure `IdentDRs(start, Node, K, T)`

```

1  maxDR = [0, 0, 0];
2  for (i = 1; i <= K; i++) /* compute for each i-combination */
3  for (f = start; f <= i; f++) /* start from each node */
4  flag = true;
5  for (j = f; j < size(Node.Children); j+i)
6  if Distance(Node, i, j) <= T then
7  if flag=true then curDR = [i, j, 2*i];
8  flag = false;
9  else curDR[3] = curDR[3] + i;
10 else if flag = false then Exit-inner-loop;
11 if (maxDR[3] < curDR[3]) and
12 (maxDR[2] = 0 or (curDR[2] <= maxDR[2])) then
13 maxDR = curDR;
14 if (maxDR[3] != 0) and
15 (maxDR[2] + maxDR[3] - 1 != size(Node.Children)) then
16 return {maxDR} ∪
    IdentDRs(maxDR[2] + maxDR[3], Node, K, T)
16 return ∅;

```

Figure 9. Identifying data regions below a node.

Finally, procedure `UnCoveredDRs` is given in Figure 10. $tempDiffDRs$ stores those data regions in $Child.DRs$ that are not covered by any data regions of $Node$.

Procedure `UnCoveredDRs(Node, Child)`

```

1  tempDiffDRs = ∅;
2  for each data region DR in Child.DRs do
3  if DR not covered by any region in Node.DRs then
4  tempDiffDRs = tempDiffDRs ∪ {DR}
5  return tempDiffDRs

```

Figure 10: The `UnCoveredDRs` procedure

Assume that the total number of nodes in the tag tree is N , the complexity of FindDRs is $O(NK^2)$. Since K is normally very small. Thus, the computation requirement of the algorithm is low.

2.3 Identify Data Records

After all data regions and their generalized nodes are found from a page, we can identify data records in each region. As noted earlier, a generalized node may not be a data record containing a single object because procedure UnCoveredDR reports higher level data regions. The actual records may be at a lower level, i.e., a generalized node may contain one or more data records.

Figure 11 shows a data region that contains two table rows (1 and 2). Row 1 and row 2 have been identified as generalized nodes. However, they are not individual data records. Each row actually contains two data (objects) records in the two cells.

row 1	Object 1	Object 2
row 2	Object 3	Object 4

Figure 11: Each row with more than one data record

To find data records from each generalized node in a data region, the following constraint is useful: *If a generalized node contains two or more data records, these data records must be similar in terms of their tag strings.* This constraint is clear because we assume that a data region contains descriptions of similar objects.

Identifying data records from each generalized node is relatively easy because they are nodes (together with their sub-trees) at the same level as the generalized node, or nodes at a lower level of the tag tree. Our experiments show that we only need to go down one level to check if data records are there. If not, the data records are at the same level as the generalized node. This is done based on the above constraint as all string comparisons have been done. This step, however, needs heuristic knowledge of how people present data objects (see [11] for details). Our method for finding non-contiguous object descriptions is also given in [11].

3. EXPERIMENT RESULTS

We evaluate our system (MDR) and compare it with two state-of-the-art systems, OMINI [2], and IEPAD [3]. Both systems are publicly available (OMINI: <http://disl.cc.gatech.edu/Omini/>, IEPAD: <http://140.115.155.99>). The results are given in Table 1. Below, we first describe some experimental settings.

Experiment Web pages: We use the whole set of 18 pages from OMINI’s homepage. Since most Web pages change frequently, three pages in OMINI did not contain any regularly data records when we performed our experiments. Thus, these 3 pages are not included. We also used a large number of other pages from a wide range of domains, books, travel, software, auctions, jobs, electronic products, shopping, and search engine results.

Edit distance threshold: We used a number of training pages (not used in testing) in building our systems and in selecting the edit distance threshold, which is decided to be 0.3.

Evaluation measures: We use the standard measures of precision and recall to evaluate the results from different systems.

Experimental results: We now discuss the results in Table 1.

Columns 1 and 2: Column 1 gives the id number of each experiment or Web page. The first 15 pages are from OMINI.

Column 2 gives the URL of each page.

Column 3: It gives the number of data records contained in each

page. These are those obvious data records of the page. They do not include navigation areas, which can have patterns as well. Since OMINI tries to identify the main objects in the page, it does not give navigation areas or other smaller regions. However, both IEPAD and MDR are able to report such regions if they exist. Although it may be possible to remove them by using some heuristics, we choose to keep them because they may be useful to some users.

Column 4: It shows the number of data records found by our system MDR. It gives perfect results for all pages except one, paper 44. For this page, MDR lost one data record.

Columns 5, 6 and 7: Column 5 shows the number of correct data records found by OMINI. Column 6 gives the total number of data records (which may not be correct) found by OMINI.

Column 7 gives some remarks about the problems with OMINI. Columns 8, 9 and 10: They give the three corresponding results of IEPAD for each page. IEPAD often produces a large number of rules to extract information from data records. We tried every rule and present the best result in column 8 for each page.

The last two rows of the table give the total number of data records in each column, the recall and the precision of each system. The precision and the recall are computed based on the total number of data records found in all pages by each system.

Before further discussing the experimental results, we first explain the problem descriptions used in the table:

all-in-n (m-in-n) with noise: It means that all (or m) data records are identified as n data records ($m > n$). “with noise” means that some items in the data records are not part of the data records.

n-err.: It means that n (extra) incorrect data records are found.

miss n objects: n correct data records are not identified.

split into n: This means that correct data records are split into n smaller ones, i.e., a data record is not found as one but a few.

none-found: None of the correct data records is found.

all-miss-info (n-miss-info): This means that all (or n) data records found by the system with some parts missing.

The following summarizes the experimental results in Table 1.

1. Our system MDR is able to give perfect results for every page except for page 44. For this page, one data record is not found because it is too dissimilar to its neighbors. From the last two rows, we can see that MDR has a 99.8% recall and 100% precision. Both OMINI and IEPAD only have a recall of 39%.
2. In columns 7 and 10, those cells that do not contain remarks show that the system finds all data records correctly. OMINI only gives perfect results for 6 pages out of 46, while IEPAD gives perfect results for only 14 pages. Our MDR system is able to give perfect results for all the pages except page 44.
3. In column 7, we see that in 20 pages, many data records are identified together as one by OMINI and also include some noisy items. This clearly shows the serious weakness of OMINI’s tag based heuristic approach.
4. Both OMINI and IEPAD are unable to find non-contiguous structures. Such cases occur in pages 1, 4 and 36.

Execution time: On a Pentium 4 PC 1.4GHz and 512 MB RAM, the execution time for each page is always less than 0.5 second.

4. CONCLUSION

This paper proposed a novel and effective technique to mine data records in a Web page. The algorithm is also able to discover non-contiguous data records, which cannot be handled by existing techniques. Experimental results show that the new method outperforms two existing state-of-the-art systems dramatically.

	URL	Obj.	MDR	OMINI			IEPAD		
				corr.	found	remark	corr.	found	remark
1	http://www.bookbuyer.com	4	4	2	4	all-miss-info	4	5	all-miss-info
2	http://www.powells.com	4	4	4	5	1 err.	0	0	none-found
3	http://www.barnesandnoble.com	4	4	0	5	all-in-1 with noise	0	7	none-found
4	http://www.codysbooks.com	6	6	0	3	all-in-1 with noise	6	7	all-miss-info
5	http://www.bookpool.com	25	25	25	26	1 err.	0	12	none-found
6	http://www.borders.com	25	25	25	25		14	14	miss 9 objects
7	http://www.alphabestreet.infront.co.uk	10	10	0	8	none-found	10	10	
8	http://www.ebay.com	7	7	0	1	all-in-1 with noise	7	7	
9	http://auctions.yahoo.com	6	6	0	3	all-in-1 with noise	6	7	1 err, all-miss-info
10	http://www.drugstore.com	8	8	0	0	none-found	7	7	miss 1 object
11	http://www.epicurious.com	3	3	0	0	none-found	0	12	none-found
12	http://www.mymenus.com	6	6	0	2	none-found	0	6	none-found
13	http://www.cooking.com	11	11	0	3	none-found	9	14	2 split into 5
14	http://www.eve.com/	9	9	0	2	all-in-1 with noise	9	9	
15	http://www.etoys.com	5	5	4	4	miss 1 object	5	5	all-miss-info
16	http://www.tourvacationstogo.com	70	70	70	70		0	5	none-found
17	http://www.tourturkey.com	6	6	5	6	1 err.	0	0	none-found
18	http://www.asiatravel.com	18	18	0	4	all-in-1 with noise	15	15	all-miss-info
19	http://www.mapquest.com	2	2	0	4	all-in-1 with noise	0	5	none-found
20	http://www.travelocity.com	5	5	0	7	all-in-1 with noise	5	5	
21	http://www.ubid.com	22	22	0	2	all-in-1 with noise	13	27	extra 14 wrong
22	http://www.grijns.net/	62	62	0	14	all-in-12 with noise	5	5	miss 57 object
23	http://journeys.20m.com	8	8	3	5	miss 5 objects	8	10	extra 2 wrong
24	http://www.softwareoutlet.com	9	9	0	3	all-in-1 with noise	8	9	extra 1 wrong
25	http://qualityinks.com/index.php	66	66	0	22	all-in-22 with noise	0	0	none-found
26	http://www.nothingbutsoftware.com	17	17	14	17	3-in-1 with noise	14	14	
27	http://www.newegg.com	12	12	0	5	6-in-3 with noise	6	6	
28	http://chemstore.cambridgesoft.com	5	5	5	5		5	5	
29	http://www.godaddy.com	4	4	0	2	all-in-1 with noise	4	11	7 err, all-miss-info
30	http://www.compusa.com	8	8	0	3	all-in-1 with noise	8	8	
31	http://www.radioshack.com	9	9	3	4	miss 6 objects	9	9	
32	http://www.earlemu.com	4	4	4	5		0	0	none-found
33	http://www.kadybooks.com	20	20	0	50	split into 50	10	10	
34	http://www.kidsfootlocker.com	9	9	0	1	all-in-1 with noise	9	9	2-miss-info
35	http://shop.lycos.com	13	13	0	2	all-in-1 with noise	5	5	
36	http://thenew.hp.com	4	4	0	5	all-in-1 with noise	0	10	split into 10
37	http://www.dell.com	5	5	5	5		5	5	
38	http://www.circuitcity.com	4	4	0	0	none-found	0	6	none-found
39	http://www.overstock.com	3	3	3	3		0	8	split into 8
40	http://www.kodak.com	3	3	0	6	none-found	0	6	none-found
41	http://www.flipdog.com	25	25	25	28	3 err.	0	0	none-found
42	http://www.summerjobs.com	20	20	0	3	none-found	0	7	none-found
43	http://search.lycos.com	10	10	0	8	all-in-2 with noise	10	10	
44	http://www.northernlight.com	10	9	10	11	1 err.	10	10	all-miss-info
45	http://www.coolhits.com	20	20	20	21	1 err.	0	0	none-found
46	http://www.mamma.com	15	15	15	20	5 err.	15	15	
	Total	621	620	242	432		241	357	
	Recall / Precision		99.8% / 100%	39% / 56%			39% / 67%		

Table 1: Experimental results

5. REFERENCES

- [1] Baeza-Yates, R. "Algorithms for string matching: A survey." *ACM SIGIR Forum*, 23(3-4):34--58, 1989
- [2] Buttler, D., Liu, L., Pu, C. "A fully automated extraction system for the World Wide Web." *IEEE ICDCS-21*, 2001.
- [3] Chang, C-H., Lui, S-L. "IEPAD: Information extraction based on pattern discovery." *WWW-10*, 2001.
- [4] Cohen, W., Hurst, M., and Jensen, L. "A flexible learning system for wrapping tables and lists in HTML documents." *WWW-2002*, 2002.
- [5] Doorenbos, R., Etzioni, O., Weld, D. "A scalable comparison shopping agent for the World Wide Web." *Agents-97*, 1997.
- [6] Embley, D., Jiang, Y and Ng, Y. "Record-boundary discovery in Web documents." *SIGMOD-99*, 1999.
- [7] Gusfield, D. *Algorithms on strings, tree, and sequence*. 1997.
- [8] Hsu, C.-N., and Dung, M.-T. "Generating finite-state transducers for semi-structured data extraction from the Web." *Information Systems*. 23(8): 521-538, 1998.
- [9] Kushmerick, N. "Wrapper induction: efficiency and expressiveness." *Artificial Intelligence*, 118:15-68, 2000.
- [10] Lerman, K. Knoblock, C., and Minton, S. "Automatic data extraction from lists and tables in web sources." *IJCAI-01 Workshop on Adaptive Text Extraction and Mining*, 2001.
- [11] Liu, B., Grossman, R. and Zhai, Y. "Mining data records in Web pages." UIC Technical Report, 2003.
- [12] Muslea, I., Minton, S. and Knoblock, C. "A hierarchical approach to wrapper induction." *Agents-99*, 1999.