# Towards a Unified Solution: Data Record Region Detection and Segmentation*

Lidong Bing Wai Lam Yuan Gu
Department of Systems Engineering and Engineering Management
The Chinese University of Hong Kong
Shatin, Hong Kong
{ldbing, wlam, yuangu}@se.cuhk.edu.hk

## ABSTRACT

Although the task of data record extraction from Web pages has been studied extensively, yet it fails to handle many pages due to their complexity in format or layout. In this paper, we propose a unified method to tackle this task by addressing several key issues in a uniform manner. A new search structure, named as Record Segmentation Tree (RST), is designed, and several efficient search pruning strategies on the RST structure are proposed to identify the records in a given Web page. Another characteristic of our method which is significantly different from previous works is that it can effectively handle complicated and challenging data record regions. It is achieved by generating subtree groups dynamically from the RST structure during the search process. Furthermore, instead of using string edit distance or tree edit distance, we propose a token-based edit distance which takes each DOM node as a basic unit in the cost calculation. Extensive experiments are conducted on four data sets, including flat, nested, and intertwine records. The experimental results demonstrate that our method achieves higher accuracy compared with three state-of-the-art methods.

## Categories and Subject Descriptors

H.3.m [**Information Storage and Retrieval**]: Miscellaneous

## General Terms

Algorithms, Performance, Experimentation

* The work described in this paper is substantially supported by grants from the Research Grant Council of the Hong Kong Special Administrative Region, China (Project Codes: CUHK4128/07 and CUHK413510) and the Direct Grant of the Faculty of Engineering, CUHK (Project Codes: 2050442 and 2050476). This work is also affiliated with the Microsoft-CUHK Joint Laboratory for Human-centric Computing and Interface Technologies.
We also acknowledge the help received from Gengxin Miao and Weiyi Meng who provided us some details of their experiments.

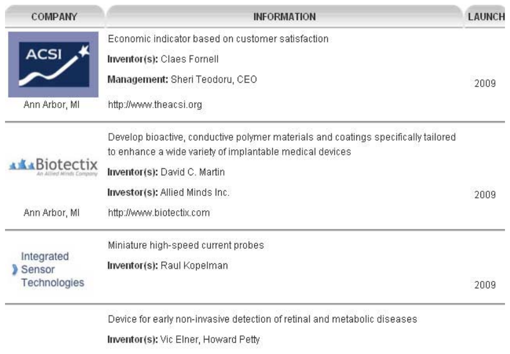
## Keywords

Web Data Record Extraction, RST Structure, Web Information Integration

## 1. INTRODUCTION

The World Wide Web has been extensively developed since its first appearance two decades ago. The explosive growth and spread of the Web have resulted in a huge information repository, but yet it is still under-utilized due to the difficulty in automated information extraction (IE) caused by data heterogeneity. The task of Web IE differs largely from the traditional IE task which aims at extracting data from basically unstructured free text. In contrast, Web IE processes Web documents (or Web pages) which are semi-structured and coded with HTML tags. Furthermore, Web sites commonly employ pre-designed templates to format and present information units, known as *data records*, which have similar attributes or fields. For example, retailing Web sites usually run a server-side program to fill products' information, retrieved from back-end databases, in a pre-defined template to generate Web pages, which are referred to as deep or dynamic Web pages. Organizations also prefer to display its information in a structured way to facilitate easy browsing, such as list of chief staff, list of breaking events, etc. These pages are known as surface or static Web pages. Therefore, the structured information on the Web is tremendously popular and very valuable for various applications such as online market intelligence [5], data mashups [1], etc.

Some existing works adopt a supervised manner to perform the Web IE task [15, 28]. They need some pre-labeled pages as input to learn some extraction rules or wrappers. Some others adopt an unsupervised manner [3, 8, 17, 20, 22], which can extract desirable data without manually prepared training data. Considering the availability of single or multiple input pages from a particular Web site, Web IE systems can also be classified into single-page based methods and multiple-page based methods. Multiple-page based methods [3, 8, 15, 28] take several pages coming from the same Web site as input, and extract the underlying template or schema automatically by analyzing the differences and similarities of these pages. Single-page based methods [17, 20, 22] do not need several similar pages from the same Web site. Therefore, they are able to handle Web sites where multiple similar pages are not available. The work introduced in this paper is a single-page based and unsupervised framework, and it can detect the *record region* (*record section*) and segment the records simultaneously under a unified model. One



| COMPANY | INFORMATION | LAUNCH |
|---|---|---|
| ACSI Ann Arbor, MI | Economic indicator based on customer satisfaction<br>**Inventor(s):** Claes Fornell<br>**Management:** Sheri Teodoru, CEO<br>http://www.theacsi.org | 2009 |
| Biotectix Ann Arbor, MI | Develop bioactive, conductive polymer materials and coatings specifically tailored to enhance a wide variety of implantable medical devices<br>**Inventor(s):** David C. Martin<br>**Investor(s):** Allied Minds Inc.<br>http://www.biotectix.com | 2009 |
| Integrated Sensor Technologies | Miniature high-speed current probes<br>**Inventor(s):** Raul Kopelman | 2009 |
| | Device for early non-invasive detection of retinal and metabolic diseases<br>**Inventor(s):** Vic Elner, Howard Petty | |

(a) A page fragment of company information.

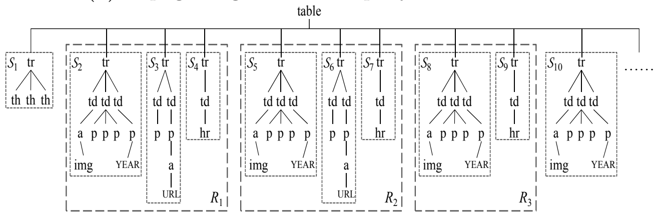


table
$S_1$ tr th th th
$S_2$ tr td td td a p p p p img YEAR
$S_3$ tr td td p p a URL
$S_4$ tr td hr
$R_1$
$S_5$ tr td td td a p p p p img YEAR
$S_6$ tr td td p p a URL
$S_7$ tr td hr
$R_2$
$S_8$ tr td td td a p p p p img YEAR
$S_9$ tr td hr
$R_3$
$S_{10}$ tr td td td a p p p p img YEAR
......

(b) DOM tree of the page in (a). $\mathcal{S}_i$ is a subtree, $R_i$ is a record.

**Figure 1: A page fragment and its DOM tree.**

example of such kind of page is given in Figure 1(a). In some existing works, further post-processings are performed such as attribute alignment [26, 30] and labeling [24, 33]. These post-processings make the extracted data suitable for downstream applications.

Two basic observations on data record and record region are as follows: 1) A group of data records describing a set of similar objects are typically presented in a contiguous region, known as record region, of a page and are formatted using similar HTML tags; 2) A set of similar data records are formed by some child subtrees of the same parent node. Based on these observations, a natural way to detect record region and record boundary is to examine the similarity (or distance) between subtrees. This strategy is adopted by MDR, ViPER, and DEPTA. MDR [17] utilizes string edit distance to assess whether two adjacent subtree groups, named as *generalized nodes*, are a repetition of the same data type. ViPER [22] is another work which also calculates string edit distance of the subtree pairs to detect record region. Then it involves some visual information to segment a detected region into records. DEPTA [27] calculates tree edit distance between generalized nodes. In summary, MDR and DEPTA calculate similarity for the pairs of neighboring generalized nodes, while ViPER calculates similarity for every pair of subtrees.

The advantage of the above similarity-based approaches is that they are robust against optional fields or tags in a record, and can tackle the problem of approximate matching to identify repeating objects. However, one disadvantage is that the method of determining the granularity involved in the similarity calculation is not flexible. In a record region, records may contain different number of subtrees, as shown in Figure 1. Thus, the number of possible subtree combinations to compose records is exponential with respect to the number of subtrees. To tackle this problem, MDR and DEPTA introduce a concept known as generalized node, which contains several subtrees. The limitation of the MDR family methods [17, 26, 27] is the greedy search for the best generalized node combination, which is time consuming as noticed by other works [20, 22]. To speed up the greedy search, one inflexible constraint is introduced in their definition of record region. This constraint specifies that all generalized nodes must have the same length, i.e. the number of subtrees, in the same record region. This constraint will fail to handle some cases in which the records contain different number of subtrees. For the record region in Figure 1, MDR treats $\mathcal{S}_2 \cdots \mathcal{S}_7$ as one record region, and misses the record $R_3$. Noticing this limitation, ViPER only calculates the similarity for single subtree pairs and constructs a similarity matrix. Then with this matrix, some heuristic rules are employed to detect the record region. Another limitation of these existing methods is that they decompose record extraction into two separate steps, namely, record region detection, and record segmentation. The reason is that the calculated similarity or distance in region detection, namely, the edit distance of generalized nodes in MDR, and the similarity of pairwise subtrees in ViPER, cannot be utilized to decide record boundary directly, since the calculated information is not necessarily boundary related.

In this paper, we propose a unified method to tackle the record extraction task. A new search structure, named as **R**ecord **S**egmentation **T**ree (RST), is designed. Based on the RST structure, several key issues in record extraction are handled in a uniform manner. For example, whether the given region contains a record region, if so, from which subtree it starts, where it ends, and how to segment it into records. To obtain solutions for these issues, we develop several efficient search pruning strategies on the RST structure of a given region to identify the correct record segmentation. When a correct segmentation is successfully detected, it implies that the record region is also successfully obtained. Based on such approach, our method is able to combine the two steps in the existing works in a unified framework. An intuitive way to understand the advantage of our method is that if the region can be segmented into records having high pairwise record similarity, then we are confident that this region is a record region. At the same time, we also have high confidence on the record segmentation.

Another characteristic of our method which is significantly different from previous works is that it can effectively handle complicated and challenging data record regions such as embedded region and nested region. It is achieved by generating subtree groups dynamically from the RST structure during the search process. These groups may contain different number of subtrees which is determined by taking into account the records' features in the current region, whereas MDR and DEPTA pre-define a subtree grouping schema, i.e. generalized node, for all record regions. Furthermore, instead of using string edit distance or tree edit distance, we propose a token-based edit distance. Different tag names have different string length, thus using edit distance directly on tag name string is not suitable. In our token-based edit distance, each tag is counted as a basic unit in the edit cost calculation.

## 2. RELATED WORK

Structured data extraction from Web pages has been studied extensively. Early works on manually constructed wrappers [4, 18] were found difficult to maintain and be applied to different Web sites, because they are very labor intensive. Semi-automatic methods [2, 13, 14, 15, 16, 21, 28, 31, 32], known as wrapper induction, were proposed to tackle this problem. These methods need some labeled pages in the target domain as input to perform the induction. Thus, they still have limitation for large-scale applications. To overcome the above drawbacks, fully automatic methods have been developed. Methods such as MDR [17], DeLa [24], ViPER [22], and DEPTA [27] were designed to tackle record-level extraction task from a single input page. Our framework falls into this category, precisely, it is an automatic, record-level, and single-page based approach.

Techniques that address record extraction from a single page can be categorized into five categories: early methods based on heuristics [6, 10], repetitive pattern based methods [7, 24], similarity-based extraction methods [17, 22, 26], tag path based methods [20], and visual feature based methods [11, 19, 29]. Methods based on heuristic rules cannot be generalized well. Repetitive pattern based methods such as IEPAD [7] and DeLa [24] show some potential in solving this problem because similar templates are used in formatting the records, which make it feasible to mine some repetitive patterns as clues for locating records in the page. One limitation of such pattern mining approaches is that it is not robust against optional data and tags inserted into records.

The similarity-based approach tackles this limitation with approximate matching to identify repeating objects. MDR [17] and DEPTA [26] are such techniques, which utilize string or tree edit distance to assess whether two adjacent subtree groups are a repetition of the same data type. Our method also calculates the similarity between two groups of subtrees, but the designed RST structure makes the subtree grouping more flexible and better aligned with the record boundary. ViPER [22] is another work which computes the similarity of each pair of single subtrees to detect record region, then involves some visual perception to segment the detected regions into records.

Miao et al. investigate the tag paths in a Web page to perform record extraction [20]. They transform a DOM tree into pieces of tag paths, and cluster the paths according to the defined similarity measure to detect record regions. The limitation of this method is that it cannot take into account the record boundary information during region detection, and needs a separate step to segment records after region detection. Therefore, tag paths in the same records and paths among different records cannot be distinguished and treated differently. Inevitably, the accuracy of both steps will be affected. Furthermore, this method clusters the tag paths across the entire page, and does not consider the proximity relations of the paths. However, the same tag path may be used in different blocks of the page, even these blocks are far away from each other.

Although ViPER [22] and work by Miao et al. [20] utilize some visual information from rendered Web page to assist record segmentation, they depend on tag structure to detect record regions. In contrast, ViNTs [29] utilizes the visual information first to identify content regularity, and then combines it with tag structure regularity to generate wrappers. ViNTs cannot separate horizontally arranged records, e.g., nested records in a table, and identify multiple regions. Pure visual feature based methods include VENTex [11] and ViDE [19], and they are effective to extract records from pages with well organized visual features. Some other researchers employ pre-defined domain ontology [9] or automatically generated domain ontology [23] to assist the record extraction task.

## 3. BASIC OBSERVATIONS AND PROBLEM DEFINITION

In the Web page fragment given in Figure 1(a), the company information is organized in a "table", and each company corresponds to one data record. The table's DOM tree is given in Figure 1(b). We can see that the records share some common fields such as company description, investor, and established year. One can also notice that some records do not contain certain fields. For example, the third record does not have URL information. Furthermore, the records are formatted with similar HTML templates, and each record is composed of several rows in the table. We use $\mathcal{T}$ to denote the DOM tree given in Figure 1(b), and $\mathcal{T}$'s subtree sequence is denoted by $\mathbf{S}$, and an element in $\mathbf{S}$ is referred to as $\mathcal{S}_i$. In the DOM tree in Figure 1(b), $\mathcal{T}$ is "table", and $\mathbf{S}$ includes $\mathcal{S}_1$, $\mathcal{S}_2$, etc. $\mathcal{T}$ and $\mathcal{S}_i$ are also used to refer to the root nodes of the corresponding DOM trees. A fragment of the sequence $\mathbf{S}$ is denoted by $\mathcal{S}_{i..j}$ or $\mathcal{S}_i \cdots \mathcal{S}_j$, where $1 \leq i \leq j \leq |\mathbf{S}|$.

Considering the characteristics of data record and record region, previous works rely on two basic observations which are reviewed as follows:

Observation 1. *A group of data records describing a set of similar objects are typically presented in a particular region of a page and are formatted using similar HTML tags.*

Observation 2. *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. It is very unlikely that a data record starts from an inner node of a child subtree and ends at an inner node of another child subtree of the parent node.*

Based on these two observations, existing works separate the record extraction task into two sub-tasks, namely, record region detection and record segmentation. In our framework, we propose a unified method which can tackle these two sub-tasks simultaneously. Returning to the example in Figure 1, given the subtree sequence $\mathbf{S}$ of the table $\mathcal{T}$, data record extraction aims at identifying the sequence $\mathcal{S}_{i..j}$ $(i < j)$ and a set of separating indexes $\mathbf{b}$ in which each $b_k \in \mathbf{b}$ *s.t.* $i < b_k \leq j$. The identified sequence $\mathcal{S}_{i..j}$ is a *record region*, and it is separated into *data records* by the indexes in $\mathbf{b}$. In the above example, the record region is $\mathcal{S}_{2..|\mathbf{S}|}$, and separating indexes set $\{4, 7, 9 \cdots\}$ indicates the boundary of records in this region. Thus, we know that the records are $\mathcal{S}_{2..4}$, $\mathcal{S}_{5..7}$, $\mathcal{S}_{8..9}$, etc. It is important to notice that a record region may not start from the first subtree of $\mathcal{T}$, and the length (number of subtrees) of different records may be different. In addition, some DOM trees may contain 0 or more than one regions. If there is no record region, no subtree sequence should be identified. If there are several regions, several subsequences of $\mathbf{S}$ should be identified.

## 4. RECORD SEGMENTATION TREE

Given a subtree sequence $\mathbf{S}$, we design a new search structure, named as Record Segmentation Tree (RST), to detect possible records in this sequence. Based on this search structure, the proposed algorithm can search and identify data records in the sequence. If some data records are identified, they naturally compose record regions. Therefore, region detection and record segmentation are performed simultaneously. For the convenience of interpretation, we use the

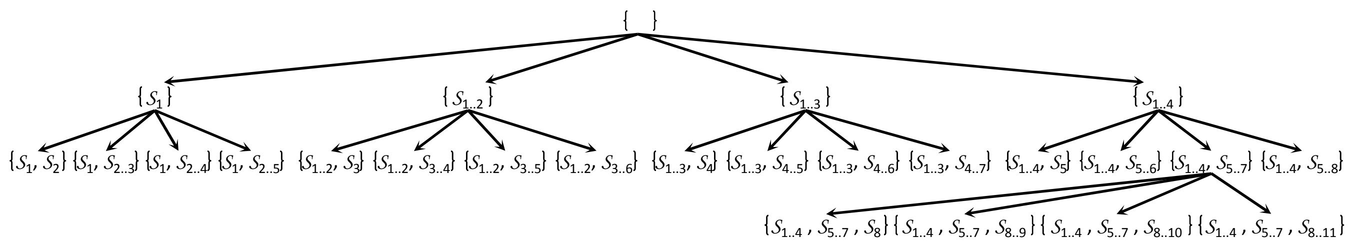

{ }
{S1}
{S1..2}
{S1..3}
{S1..4}
{S1, S2} {S1, S2..3} {S1, S2..4} {S1, S2..5}
{S1..2, S3} {S1..2, S3..4} {S1..2, S3..5} {S1..2, S3..6}
{S1..3, S4} {S1..3, S4..5} {S1..3, S4..6} {S1..3, S4..7}
{S1..4, S5} {S1..4, S5..6} {S1..4, S5..7} {S1..4, S5..8}
{S1..4 , S5..7 , S8} {S1..4 , S5..7 , S8..9} {S1..4 , S5..7 , S8..10} {S1..4 , S5..7 , S8..11}

**Figure 2: An example of basic record segmentation tree with $K = 4$.**

subsequence starting from $\mathcal{S}_1$ to illustrate RST construction, as well as the subsequent algorithms. Our framework can detect the region starting from any subtrees in $\mathbf{S}$. We assume that one record at most contains $K$ subtrees in $\mathbf{S}$.

## 4.1 Basic Record Segmentation Tree

Definition 1. ***Record segmentation tree*** *is a search tree with the following properties:*

- *Each node* $\mathbf{R}$ *represents a possible record region. The root node represents an empty region.*
- *Each* $\mathbf{R}$ *covers a prefix subsequence* $\mathcal{S}_{1..n}$ *of* $\mathbf{S}$*, referred to as* $\mathbf{S_R}$*, where* $0 \le n \le |\mathbf{S}|$*, and has a separating indexes set* $\mathbf{b}$ *which segments* $\mathcal{S}_{1..n}$ *into records. Each record of* $\mathbf{R}$ *is denoted by* $R_i$*. The root has an empty prefix subsequence, i.e.* $n = 0$*, and an empty separating indexes set.*
- *Each* $\mathbf{R}$ *with* $\mathcal{S}_{1..n}$ *and* $\mathbf{b}$ *has at most* $K$ *children. Each child of* $\mathbf{R}$ *covers* $\mathcal{S}_{1..m}$ *where* $n+1 \le m \le \min\{n+K, |\mathbf{S}|\}$*, and has a separating indexes set* $\mathbf{b} \cup \{m\}$*.*

From the above definition, it can been seen that each node is recursively defined with its parent node, and contains one more record. The record extraction task in $\mathbf{S}$ is transformed into a problem that searches a node in the RST structure which best matches with the true records in $\mathbf{S}$.

The segmentation tree with $K = 4$ is given in Figure 2. In this example, each node is labeled by its record set. For instance, the node $\mathbf{R} = \{\mathcal{S}_{1..4}, \mathcal{S}_{5..7}\}$ indicates that there are two records, namely, $R_1 = \mathcal{S}_{1..4}$, and $R_2 = \mathcal{S}_{5..7}$. The covered prefix is $\mathcal{S}_{1..7}$, and the separating indexes set is $\{4, 7\}$. $|R_i|$ denotes the length of $R_i$, and is equal to the number of subtrees it contains. For instance, $|R_1| = 4$, $|R_2| = 3$. We use $|\mathbf{R}|$ to denote the number of records in $\mathbf{R}$. $L_\mathbf{R}$ denotes the average length of the records in $\mathbf{R}$, calculated as $(\sum_{R_i \in \mathbf{R}} |R_i|)/|\mathbf{R}|$ or $|\mathbf{S_R}|/|\mathbf{R}|$.

## 4.2 Slimmed Segmentation Tree

In the same record region, it is almost impossible that the lengths of different records are significantly different, say some records contain only 1 subtree, while some others contain 5 subtrees. Based on this observation, we design a slimmed segmentation tree, in which the length of previous records is used to predict that of the next record. Suppose for a node $\mathbf{R}$ with $L_\mathbf{R} = 2$, we may think that it is impossible that a child of $\mathbf{R}$ has a new record with length 4 or more. Therefore, we only consider the children of $\mathbf{R}$ in which the new records' length is smaller than 4. Another observation is that when $L_\mathbf{R}$ is large, it is more probable that different records have a larger absolute length difference. For example, in a page in our experiment data set, the average length of the records is 7. Some records contain 4 subtrees, while some others contain 9 subtrees. Taking the above two observations into consideration, we introduce a slimmed version of segmentation tree.

Definition 2. ***Slimmed segmentation tree*** *is a subgraph structure of the basic record segmentation tree. It keeps the first two layers of basic RST. Each child of a non-root node* $\mathbf{R}$ *with prefix sequence* $\mathcal{S}_{1..n}$ *has prefix sequence* $\mathcal{S}_{1..m}$*, where* $n + (L_\mathbf{R} - \lfloor L_\mathbf{R}/2 \rfloor) \le m \le \min\{n + \min\{L_\mathbf{R} + \lceil L_\mathbf{R}/2 \rceil, K\}, |\mathbf{S}|\}$*.*

An example is given in Figure 3, in which $K$ is 5. It can be seen that in the third layer, we do not construct all possible children of a node $\mathbf{R}$ in the second layer. If $L_\mathbf{R}$ is smaller, we construct fewer number of children for it. Otherwise, we construct more.

## 4.3 Utilize RST in Record Extraction

Each node in RST is one possible segmentation of the record region starting from $\mathcal{S}_1$ in the sequence $\mathbf{S}$. According to the observation that the records in the same region are formatted using similar tags, the correct segmentation should be the node that achieves higher average pairwise record similarity. If we cannot find a node with pairwise record similarity greater than a pre-defined threshold, we may conclude that no record region exists starting from $\mathcal{S}_1$.

Precisely, given a DOM tree $\mathcal{T}$ and its subtree sequence $\mathbf{S}$, record extraction with the RST structure of $\mathbf{S}$ aims at finding a node $\mathbf{R}^*$ such that:

$$\mathbf{R}^* = \operatorname*{argmax}_{\mathbf{R} \in \{\mathbf{R} | Q(\mathbf{R}) \ge \theta\}} |\mathbf{R}|, \tag{1}$$

where $\theta$ is a pre-defined threshold. $Q(\cdot)$ is the quality function of an RST node $\mathbf{R}$, which is defined as the average pairwise record similarity of records in $\mathbf{R}$:

$$Q(\mathbf{R}) = \frac{\sum_{R_i, R_j \in \mathbf{R}\ s.t.\ i<j} sim(R_i, R_j)}{|\mathbf{R}|(|\mathbf{R}| - 1)/2}, \tag{2}$$

where $sim$ is a similarity function between two records. Let $\mathbb{R}^*$ denote $\{\mathbf{R} | Q(\mathbf{R}) \ge \theta\}$, each element in $\mathbb{R}^*$ is a node whose quality is not less than $\theta$. Among these nodes in $\mathbb{R}^*$, the one with the maximum number of records is determined as the correct record region $\mathbf{R}^*$ starting from $\mathcal{S}_1$ in $\mathbf{S}$. If more than one nodes have the maximum number of records, we select the one with the best quality as $\mathbf{R}^*$. If $\mathbb{R}^* = \emptyset$, it means that no record region exists starting from $\mathcal{S}_1$.

For un-slimmed RST, when we expand it to layer $|\mathbf{S}|/K$, each inner node has $K$ children. Thus, the total number of nodes in layer $|\mathbf{S}|/K$ is $K^{|\mathbf{S}|/K}$. In the slimmed version, suppose each inner node has $\hat{K}$ children on average, this number is $\hat{K}^{|\mathbf{S}|/K}$. Usually, in a possible record region, $|\mathbf{S}|$ is much larger than $K$. Constructing and searching such a segmentation tree is extremely time and space consuming because of its exponential size. Therefore, some search pruning strategies have been developed to allow efficient utilization of the RST structure.

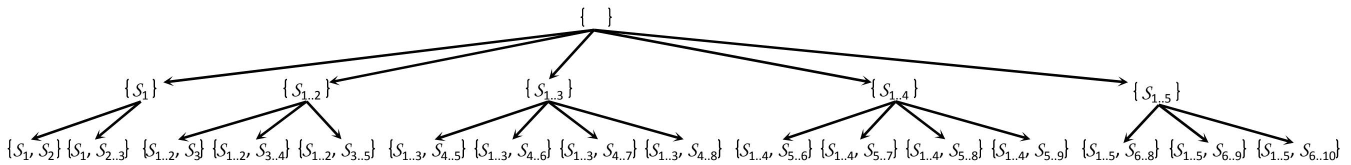


**Figure 3: An example of slimmed segmentation tree with $K = 5$.**

## 5. SEARCH PRUNING STRATEGIES

In this subsection, we introduce a threshold-based top $k$ search that can prune the RST significantly, and reduce the complexity to $O(|\mathbf{S}|^2)$ without considering pairwise similarity computation of subtrees. Furthermore, instead of calculating the similarity of all record pairs in Equation 2, we may only check the similarity of a record and its nearest previous neighbors. Accordingly, the time complexity is further reduced to $O(|\mathbf{S}|)$.

### 5.1 Threshold-Based Top k Search

#### 5.1.1 *Initialization*

In the beginning of RST construction, we have no idea about the record length in the given subtree sequence. The only information on hand is that each record has at most $K$ subtrees. To attain a better starting, we fully expand the RST in the first 2 layers as shown in Figure 2. Each non-leaf node has $K$ children in these layers. Then we have $K^2$ initial candidate nodes, denoted by $\tilde{\mathbb{R}}$, for future expansion and search. Note that $K$ is very small, and the full expansion in the beginning will not cause much increasing of time complexity.

Before proceeding to the next layer of RST, we prune the candidates in $\tilde{\mathbb{R}}$ according to their quality, which is defined in Equation 2. For a particular node $\mathbf{R}$ in $\tilde{\mathbb{R}}$, it will be pruned if $Q(\mathbf{R}) < \theta'$. The meaning of $\theta'$ will be clear later. After the nodes with lower quality are pruned, if the number of retained nodes is more than a pre-defined threshold $k$, only the top $k$ nodes with the best quality are retained, and denoted by $\mathbb{R}$. For example in Figure 2, the node $\{\mathcal{S}_1, \mathcal{S}_{2..5}\}$ is very likely to be pruned. If $\mathbb{R} = \emptyset$, then no region starting from $\mathcal{S}_1$ exists in $\mathbf{S}$.

#### 5.1.2 *Pruning Search*

Suppose $\mathbf{R}$ is a node in $\mathbb{R}$, the last record in $\mathbf{R}$ is $\mathcal{S}_{i..n}$. We construct some children of $\mathbf{R}$, namely, $\mathbf{R} \cup \{\mathcal{S}_{n+1}\}, \mathbf{R} \cup \{\mathcal{S}_{n+1..n+2}\}, \cdots, \mathbf{R} \cup \{\mathcal{S}_{n+1..n+K}\}$. Then the similarity between newly generated records and the existing ones in $\mathbf{R}$ are calculated. Precisely, for each new record $\mathcal{S}_{n+1..m}$ $(n+1 \leq m \leq n+K)$ and each $R \in \mathbf{R}$, $sim(\mathcal{S}_{n+1..m}, R)$ is calculated. Then, the quality of $\mathcal{S}_{n+1..m}$ is defined as:

$$Q(\mathcal{S}_{n+1..m}) = \frac{\sum_{R \in \mathbf{R}} sim(\mathcal{S}_{n+1..m}, R)}{|\mathbf{R}|}. \qquad (3)$$

In the same way, each $\mathbf{R}$ in $\mathbb{R}$ is expanded, and the quality of each new record is calculated.

After the RST structure is expanded one more layer, a new pruning strategy different from that for $\tilde{\mathbb{R}}$ is adopted. For a new RST node $\mathbf{R} \cup \{\mathcal{S}_{n+1..m}\}$, if $Q(\mathcal{S}_{n+1..m}) \geq \theta'$ and $Q(\mathbf{R} \cup \{\mathcal{S}_{n+1..m}\}) \geq \theta$, $\mathbf{R} \cup \{\mathcal{S}_{n+1..m}\}$ is retained, otherwise it is pruned, where $\theta'$ is smaller than $\theta$. The rationale behind this strategy is that some record, say $\mathcal{S}_{n+1..m}$, may have a larger difference compared with the previous records. But we assume that the difference should not be large. Precisely, $Q(\mathcal{S}_{n+1..m})$ should not be less than a looser threshold $\theta'$. If this condition is met, we continue to check the quality of $\mathbf{R} \cup \{\mathcal{S}_{n+1..m}\}$. In this way, the search procedure can overcome the problems caused by some outlier records, meanwhile, the quality of the outliers is also bounded by $\theta'$. Recall that we use $\theta'$ instead of $\theta$ in the pruning of the initial candidate nodes $\tilde{\mathbb{R}}$, it is because each element of $\tilde{\mathbb{R}}$ contains two records. For a particular $\mathbf{R}$, if no $\mathbf{R} \cup \{\mathcal{S}_{n+1..m}\}$ is retained after pruning, $\mathbf{R}$ is put into $\mathbb{R}^*$. Again, if the number of retained nodes is more than $k$, we keep the top $k$ nodes with the best quality, and construct a new $\mathbb{R}$.

The above procedure is repeated on the new $\mathbb{R}$ until it is empty or the end of $\mathbf{S}$ is reached. Then we finish building an RST structure for $\mathbf{S}$ starting from $\mathcal{S}_1$, at the same time, $\mathbb{R}^*$ is also built. Note that the RST structure needs not reach $\mathcal{S}_{|\mathbf{S}|}$, which indicates that some subtrees in the end of $\mathbf{S}$ should not be included in the record region.

#### 5.1.3 *Retrospect with Short-term Memory*

In the above expanding and pruning process, when examining the quality of a new record, all previous records are considered, as shown in Equation 3. This thorough retrospect is not only time consuming, but also unnecessary. Especially in the region with many records, the cost of such exhaustive comparison outweighs the benefit gained. We adopt a short-term memory retrospect strategy to reduce the computation workload. When a node is expanded one more layer, we only calculate the similarity between a new record and the nearest $r$ previous records. Thus, in Equation 3, the number of pairwise similarity calculations is reduced from $|\mathbf{R}|$ to $r$. And in Equation 2, the cost of evaluating one node is reduced from $|\mathbf{R}|(|\mathbf{R}|-1)/2$ to $r|\mathbf{R}|$.

### 5.2 Complexity Analysis

The computation workload of pairwise record similarity calculation is related to the number of subtrees in the involved records. Hence, we use a finer unit to analyze the complexity of utilizing RST in record extraction.

In Section 5.1, when evaluating the quality of new records with Equation 3, we need to calculate the similarity between each existing record in $\mathbf{R}$ and each new record. The computation workload is:

$$|\mathbf{R}| \sum_{1 \leq l \leq K} (L_{\mathbf{R}} l) = \frac{K(K+1)|\mathbf{S}_{\mathbf{R}}|}{2}, \qquad (4)$$

where $L_{\mathbf{R}}$ is the average length of records in $\mathbf{R}$, $\mathbf{S}_{\mathbf{R}}$ is the subtree sequence covered by $\mathbf{R}$, and $L_{\mathbf{R}} l$ indicates the computation needed for calculating the similarity between the records having $L_{\mathbf{R}}$ and $l$ subtrees respectively. Thus, the computation workload of segmenting entire $\mathbf{S}$ is:

$$\sum_{1 \leq x \leq (\frac{|\mathbf{S}|}{L_{\mathbf{R}}} - 1)} (x \sum_{1 \leq l \leq K} (L_{\mathbf{R}} l)) = \frac{K(K+1)(\frac{|\mathbf{S}|}{L_{\mathbf{R}}} - 1)|\mathbf{S}|}{4}, \qquad (5)$$

In the top $k$ strategy, the overall computation needed is $kK(K+1)(|\mathbf{S}|/L_{\mathbf{R}}-1)|\mathbf{S}|/4$. In our framework, the method of calculating pairwise record similarity is a variant of edit distance. Due to the dynamic programming employed in edit distance, when calculating the distance matrix between two strings, the existing matrix between their prefix strings can be reused. In Equation 3, when calculating the similarity between $\mathcal{S}_{n+1..m}$ and an existing record $R$, the distance matrix between $\mathcal{S}_{n+1..m-1}$ and $R$ can be reused. Consequently, the above workload can be further reduced to $k(K+1)(|\mathbf{S}|/L_{\mathbf{R}}-1)|\mathbf{S}|/4$. The value of $k$ is usually smaller than $K$, and both $k$ and $K$ are small and can be treated as constants. Thus, the overall time complexity is $O(|\mathbf{S}|^2)$.

Under the strategy of short-term memory retrospect, the workload in Equation 5 becomes:

$$(\frac{|\mathbf{S}|}{L_{\mathbf{R}}}-1)r\sum_{1\leq l\leq K}(L_{\mathbf{R}}l)=\frac{rK(K+1)|\mathbf{S}|}{2}, \qquad (6)$$

where $r$ is the number of records retrospected from current layer. Similarly, the overall complexity can be further reduced to $r(K+1)|\mathbf{S}|/2$, which can be regarded as $O(|\mathbf{S}|)$.

For each sequence $\mathbf{S}$, MDR [17] has time complexity of $O(|\mathbf{S}|)$ to calculate the similarity among the same length generalized node. In MDR, only the adjacent generalized node pairs are considered in similarity calculation, which is similar to our retrospect strategy with $r = 1$. ViPER [22] has time complexity of $O(|\mathbf{S}|^2)$ to construct the upper triangular similarity matrix of all subtree pairs. Note that the above time complexity is only the part needed by MDR or ViPER for record region detection, and they need another step to segment the detected region into records.

## 5.3 Composite Node Pruning

Suppose each subtree in $\mathbf{S}$ is one record, all RST nodes with the form $\{\mathcal{S}_{1..i}, \mathcal{S}_{i+1..2i}, \mathcal{S}_{2i+1..3i}, \cdots\}$ $(1 \leq i \leq K)$ have quite high quality. The reason is that the combination of $i$ records is similar to that of the other $i$ records. In some other cases, this kind of combination may even achieve higher quality than the correct record segmentation. For example, there are four records in a region $\mathbf{R} = \{\mathcal{S}'_1\mathcal{S}_2, \mathcal{S}'_3, \mathcal{S}'_4\mathcal{S}_5, \mathcal{S}'_6\}$, where the primed subtrees are matched ones in different records. The node $\mathbf{R}^c = \{\mathcal{S}'_1\mathcal{S}_2\mathcal{S}'_3, \mathcal{S}'_4\mathcal{S}_5\mathcal{S}'_6\}$ has a higher quality than $\mathbf{R}$. We call $\mathbf{R}^c$ a composite node of $\mathbf{R}$. Composite nodes will increase the computation workload and involve noise during RST search. To tackle this problem, after a period of search, say when all nodes in $\mathbb{R}$ exceed $\mathcal{S}_n$ in $\mathbf{S}$, we check the relation between the nodes in $\mathbb{R}$. If one node is a composite node of any other node, it is pruned. Note that for the node involving subtrees after $\mathcal{S}_n$, we only consider its records before $\mathcal{S}_n$ in the detection of composite node.

## 5.4 More Challenging Record Region Discussion

In this subsection, we discuss how several kinds of more challenging record regions can be handled with the proposed RST structure and search strategy.

### 5.4.1 *Embedded Region and Non-continuous Region*

Record region in $\mathbf{S}$ may not start from $\mathcal{S}_1$ because of the existence of header information, one example and its DOM tree are given in Figures 4(a) and 4(d). In the same way, the region also may not end at $\mathcal{S}_{|\mathbf{S}|}$. We name this kind of region as *embedded region*. The way to detect embedded regions in our framework is straightforward. In the beginning, we start from $\mathcal{S}_1$, and generate an initial set $\tilde{\mathbb{R}}_{\mathcal{S}_1}$. If all elements in $\tilde{\mathbb{R}}_{\mathcal{S}_1}$ are pruned, we move to $\mathcal{S}_2$, and repeat the above procedure. In this way, the header subtrees in the beginning are excluded. If $\mathbb{R} = \emptyset$ before coming to $\mathcal{S}_{|\mathbf{S}|}$, it means that there is some footer information which should not be treated as part of any record.

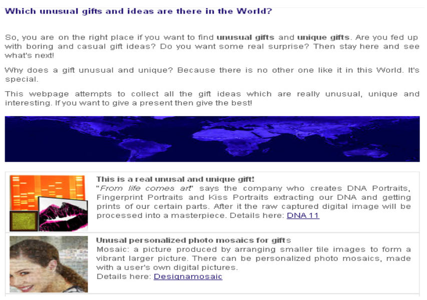
In some other cases, there may be more than one regions in $\mathbf{S}$. For example, in an on-line shopping Web page, $\mathbf{S}$ may contain two subsequences introducing new arrival products and featured products respectively. And they are separated by some other information. We name this kind of region as *non-continuous region*. The above procedure for dealing with embedded regions can also be applied to non-continuous region easily. After finishing one region detection, the detection algorithm will move on to the sequence of the remaining subtrees if there are some, and perform the search procedure again to detect the second region.

### 5.4.2 *Nested Region*

In some Web pages, the records are formatted in a nested manner. For example, the region formatted with <table>, each record is packed with one <td>, each <tr> has several <td>s. The region with <table> as root node is know as *nested region*. In our framework, with a top-down manner to scan the DOM tree, <table> is detected as a record region with <tr>s as records. Then, each <tr> is detected as a region with <td>s as records. After the <tr>s are detected as regions, along with the fact that these <tr>s have been detected as records of the same root node <table> in the preceding detection, we may postulate that the <table> is a nested region. To inspect our postulation, we compare the records, i.e., <td>s, from different <tr>s, and check whether they are similar. If so, we conclude the node <table> is a nested region. There are several other issues such as the orphan record detection in the last "row" will not be discussed here due to the limited space. Note that the above "table" is used to demonstrate the detecting process, our detection method does not rely on any particular tag.

### 5.4.3 *Intertwine Records*

*Intertwine record*, also known as *non-continuous record* in DEPTA [27], refers to the record whose attributes intertwine together with other records' attributes. One example and its DOM tree are given in Figures 4(b) and 4(e). Each record has 3 attributes, namely, image, title, and price, and these attributes scattered in 3 different subtrees (<tr>s). In our framework, the subtree sequence of <tr>s in the table is first passed to the record detection algorithm, and each successive non-overlapping 3-subtree segment is detected as one record. After that, each <tr> is passed to the detection algorithm, and detected as a region with each <td> as one record. Up to now, the above procedure resembles that of nested region detection. We continue to check the similarity between records in neighboring <tr>s, and they are found dissimilar to each other. Thus, we conclude that a intertwine region is detected, and the correct records can be generated by reassembling the detected regions. In this example, three regions (image 1, image 2), (title 1, title 2) and (price 1, price 2) are reassembled to generate records (image 1, title 1, price 1) and (image 2, title 2, price 2).



Which unusual gifts and ideas are there in the World?

So, you are on the right place if you want to find **unusual gifts** and **unique gifts**. Are you fed up with boring and casual gift ideas? Do you want some real surprise? Then stay here and see what's next!

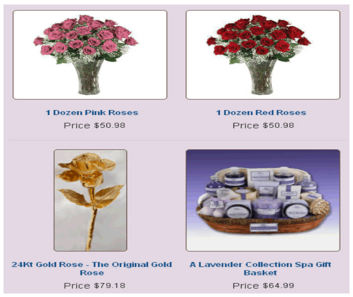
Why does a gift unusual and unique? Because there is no other one like it in this World. It's special.

This webpage attempts to collect all the gift ideas which are really unusual, unique and interesting. If you want to give a present then give the best!

**This is a real unusal and unique gift!**
"*From life comes art*" says the company who creates DNA Portraits, Fingerprint Portraits and Kiss Portraits extracting our DNA and getting prints of our certain parts. After it the raw captured digital image will be processed into a masterpiece. Details here: DNA 11

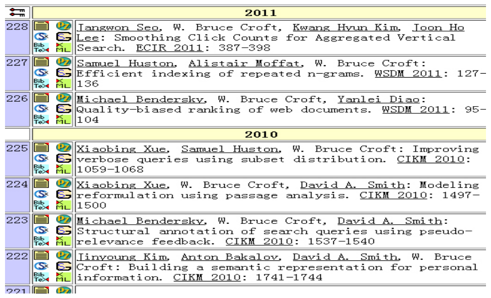
**Unusal personalized photo mosaics for gifts**
Mosaic: a picture produced by arranging smaller tile images to form a vibrant larger picture. There can be personalized photo mosaics, made with a user's own digital pictures.
Details here: Designamosaic

(a) A page fragment of embedded region.



| | |
|---|---|
| 1 Dozen Pink Roses<br>Price $50.98 | 1 Dozen Red Roses<br>Price $50.98 |
| 24Kt Gold Rose - The Original Gold Rose<br>Price $79.18 | A Lavender Collection Spa Gift Basket<br>Price $64.99 |

(b) A page fragment of intertwine region.



**2011**

228 Jangwon Seo, W. Bruce Croft, Kwang Hyun Kim, Joon Ho Lee: Smoothing Click Counts for Aggregated Vertical Search. ECIR 2011: 387-398

227 Samuel Huston, Alistair Moffat, W. Bruce Croft: Efficient indexing of repeated n-grams. WSDM 2011: 127-136

226 Michael Bendersky, W. Bruce Croft, Yanlei Diao: Quality-biased ranking of web documents. WSDM 2011: 95-104

**2010**

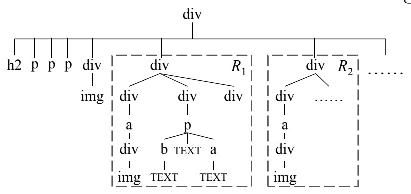
225 Xiaobing Xue, Samuel Huston, W. Bruce Croft: Improving verbose queries using subset distribution. CIKM 2010: 1059-1068

224 Xiaobing Xue, W. Bruce Croft, David A. Smith: Modeling reformulation using passage analysis. CIKM 2010: 1497-1500

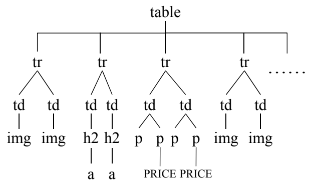
223 Michael Bendersky, W. Bruce Croft, David A. Smith: Structural annotation of search queries using pseudo-relevance feedback. CIKM 2010: 1537-1540

222 Jinyoung Kim, Anton Bakalov, David A. Smith, W. Bruce Croft: Building a semantic representation for personal information. CIKM 2010: 1741-1744

(c) A page fragment of noisy subtree region.



(d) DOM tree of (a)



(e) DOM tree of (b)

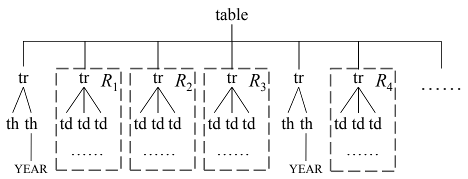


table
tr R1 R2 R3 tr R4 ......
th th td td td td td td td td td th th td td td
YEAR ...... ...... ...... YEAR ......

(f) DOM tree of (c)

**Figure 4: Several kinds of more challenging record regions.**

### 5.4.4 *Noise Subtree Exclusion*

In Figures 4(c) and 4(f), a publication list is separated into different sections according to the published year. It can be regarded as a non-continuous region. Theoretically, with proper setting of thresholds, we can expect that publications in 2011 will be detected as the first region, and publications in 2010 will be detected as the second region. However, it is also possible that the year row 2011 fuses into record 228, and the year row 2010 fuses into record 225. If a record contains such noise subtrees, the similarity between this record and its context records becomes lower. Meanwhile, the number of subtrees in this record is more than that in others. Using these observations as clues, it is not difficult to exclude noise subtrees from records. Due to space limitation, we only outline the idea and omit the details. Basically, if some subtrees in the first record are detected as noise and excluded, we need to reexamine this record segmentation. The reason is that the polluted first record may lead to wrong segmentation of the region.

## 6. RECORD SIMILARITY MEASURE

In this section, we discuss the method used for measuring the similarity between two records (sequences of subtrees), in sequence **S**. There are mainly two existing approaches. One is string edit distance based [17, 22], the other is tree edit distance based [27]. In string edit distance based method, each subtree is encoded with a string which is obtained by traversing the subtree in pre-order, and appending the name of visited tag node to the string. Due to the fact that names of different tags may have different length, using the string of tag name directly is not suitable. In spite of this limitation, string edit distance can tackle repetitive fields properly by tandem repeats detection [22], as well as optional fields. In [27], top-down distance, which is a restricted version of tree edit distance, is employed to measure the distance between two subtrees. Its problem is that any crossing layers' operation is not permitted, making it unable to handle optional tags. On the other hand, the tree edit distance can overcome the limitation in string edit distance since it considers each DOM node as an inseparable unit. To adopt their good points and avoid the shortcomings, we propose a token-based edit distance method.

**Table 1: Types of text node.**

| Type Name | Meaning | Priority |
|---|---|---|
| EMAIL | An email address | 1 |
| URL | A URL string | 2 |
| PRICE | Digital number with a currency symbol | 3 |
| TIME | Time in predefined format, such as hh:mm:ss | 4 |
| DATE | Date in predefined format, such as yyyy/mm/dd | 5 |
| YEAR | Four digits, arrange from 1900 to 2050 | 6 |
| TEXT | Other kinds of text | 7 |

### 6.1 Encoding Subtree with Tokens

Aiming at fixing the ill-formatted Web pages, some existing works [26, 27] introduce visual information from rendered Web page into DOM tree building. To avoid this time consuming rendering operation, we employ an HTML cleansing package, namely, HtmlCleaner[1], to clean the Web pages. Then the DOM structure of the cleaned page is built.

In the DOM tree, we have two kinds of nodes, namely, tag node and text node. Each tag node has a name such as "tr", "div", etc. A piece of visible text between a pair of ">" and "<" is regarded as a basic text unit, and normalized to a text node. Table 1 shows the types of text nodes defined in our framework. The priority indicates the order in which different types are attempted when normalizing a piece of text. Each subtree in the DOM is encoded with the sequence of node names in it, which is obtained by traversing the tree in pre-order. Each node name in the sequence is called one *token*, and it is an inseparable unit in similarity calculation.

### 6.2 Tandem Repeat Detection and Distance-based Measure

A typical characteristic of data records is that they vary in optional or repetitive fields. For example, a book record may have discounted price or not, and one author or several. An obvious disadvantage of edit distance computation is that repetitive and optional fields increase the edit cost.

[1] http://htmlcleaner.sourceforge.net/

As noticed by ViPER [22], similarity threshold is suitable to solve the problem caused by optional fields, but not suitable for repetitive fields. ViPER identifies *tandem repeats* in two strings before distance calculation, and allows zero cost for deletion and insertion inside additional repetitions.

In our token-based edit distance scenario, the above idea is also applicable. We implement the algorithm proposed by Gusfield et al. [12] to detect the tandem repeats. This algorithm utilizes suffix tree structure to identify the tandem repeats which are not longer than $z$ in a sequence of length $n$ in $O(n + z)$ time. The difference is that the basic unit in our sequences is token (node name), not single character. Thus we need to perform token comparison instead of character comparison in the detection of tandem repeats. Furthermore, if the token sequence is originated from several subtrees, it is constrained that tandem repeats are only detected in the token sequence of each single subtree. We refine the tandem repeats based edit distance proposed in ViPER [22] and make it suitable for the token scenario in our framework. For the details of tandem repeat detection and its application in edit distance, we would like to direct the readers to [12] and [22]. Then the calculated distance is normalized into the interval [0, 1], and the negative value of the normalized distance is added by 1. Thus, we obtain the record similarity measure used in our framework.

## 7. EXPERIMENTS

For evaluating the performance of our method and conducting comparison with existing methods, we attempt to use existing data sets and available implementation of existing methods. MDR implementation is available[2], which is able to deal with flat, nested, and intertwine records. Therefore, it was employed as the comparison baseline for all experiments. We used two existing data sets for testing the performance of our method on flat data record extraction. Besides MDR, we also compare with two other existing methods, namely, TPC [20], and ViNTs [29], whose experimental results are available on these data sets. Because no existing data set for nested or intertwine records is available, we collected one data set for each of them, and only compare our method with MDR. With respect to evaluation metrics, we employ the commonly used precision and recall. To avoid the evaluation bias brought in by pages with large number of records, both micro average and macro average are reported.

Given a Web page, our algorithm adopts a top down manner to scan its DOM tree for detecting records. For some pages, several record regions are detected. We first filter out the regions whose record size is smaller than 3, where the size of a record is defined as the number of leaf DOM nodes it has. After that, we employ the single-linkage agglomerative hierarchical clustering algorithm to cluster the remaining regions into different clusters. The algorithm starts with each region as an initial cluster which contains its records as the elements in it. The stopping similarity threshold is set to be $\theta'$. After clustering, the cluster with the largest number of records is selected as the final result. We used a number of training pages collected separately to tune the parameter values in our framework. Finally, $\theta$ is set to 0.75, $\theta'$ is set to 0.65, $K$ is set to 10, both $k$ and $r$ are set to 5.

[2] http://www.cs.uic.edu/~liub/WebDataExtraction/

**Table 2: Experimental results on TB1.**

| | | Ground | TP | FP | P-mi | R-mi | P-ma | R-ma |
|---|---|---|---|---|---|---|---|---|
| ALL | Our | 864 | 855 | 21 | .976 | .990 | .968 | .974 |
| | MDR | 864 | 553 | 48 | .920 | .640 | .594 | .610 |
| ALL/wrong | Our | 851 | 843 | 16 | .981 | .991 | .980 | .977 |
| | MDR | 851 | 553 | 48 | .920 | .650 | .620 | .636 |
| TPC used | Our | 781 | 778 | 16 | .980 | .996 | .977 | .984 |
| | TPC | 781 | NA | NA | NA | NA | .904 | .931 |

**Table 3: Experimental results on TB2.**

| | | Ground | TP | FP | P-mi | R-mi | P-ma | R-ma |
|---|---|---|---|---|---|---|---|---|
| ALL | Our | 968 | 953 | 10 | .990 | .985 | .965 | .958 |
| | MDR | 968 | 722 | 66 | .916 | .746 | .693 | .734 |
| | ViNTs | 968 | 934 | 11 | .988 | .965 | .973 | .950 |
| ALL/wrong | Our | 863 | 849 | 10 | .988 | .984 | .961 | .956 |
| | MDR | 863 | 722 | 66 | .916 | .837 | .773 | .819 |
| | ViNTs | 863 | 830 | 11 | .987 | .962 | .970 | .945 |

### 7.1 Flat Record Extraction

Our first data set is the testbed collected by Yamada et al. [25] which is available at http://daisen.cc.kyushu-u.ac.jp/TBDW/. This testbed was also used by other works such as ViPER and TPC [20]. The testbed data has 253 Web pages from 51 Web sites randomly drawn from 114,540 Web pages with search forms. The data records in this testbed are manually labeled by the collectors, and results are also available online together with the data set. In our experiment, three sites are excluded because of nested records, garbled code, or ambiguous record annotation. Thus, we use the remaining 48 sites including 238 pages in total. TPC also conducted experiment on this testbed data set. The authors kindly provided us the Web site IDs in the subset they used, which contains 43 sites out of 48 sites we use. Therefore, without implementing their method, we can still conduct a fair comparison. Our second data set is the data set 3 used in ViNTs [29] which is available at http://www.data.binghamton.edu:8080/vints/ along with the details of ViNTs' performance on each page. Thus, we can conduct a comparison with ViNTs in more detail. This data set is originated from Omini [6] testbed collected by Buttler et al., which consists of more than 2,000 Web pages collected 50 Web sites. ViNTs took one random page per Web site to construct its data set 3. In our experiment, we exclude 2 pages because of the ambiguity on record annotation. Thus, our second data set contains 48 pages. The above two data sets are referred to as TB1 and TB2 respectively. We run MDR on both data sets with the default similarity threshold 60%, and extract the records reported. MDR could not produce output for 2 Web sites in TB1 and 5 pages in TB2 because the MDR program terminated abnormally. We report both results with and without these pages.

The experimental results on TB1 and TB2 are given in Tables 2 and 3 respectively. "ALL" refers to the entire page set used, "ALL/wrong" refers to the subset without the pages on which MDR program terminated abnormally, and "TPC used" refers to the subset used by the TPC method. "Ground" denotes the number of ground truth records, "TP" denotes the number of true positive given by a method, and "FP" denotes the number of false positive. "P-mi", "P-ma", "R-mi" and "R-ma" are the micro-averaged and macro-averaged precision and recall values respectively. "NA" means no corresponding result reported in TPC [20]. In P-ma calculation, if both TP and FP are 0 for a particular page, its precision is set to be 0.

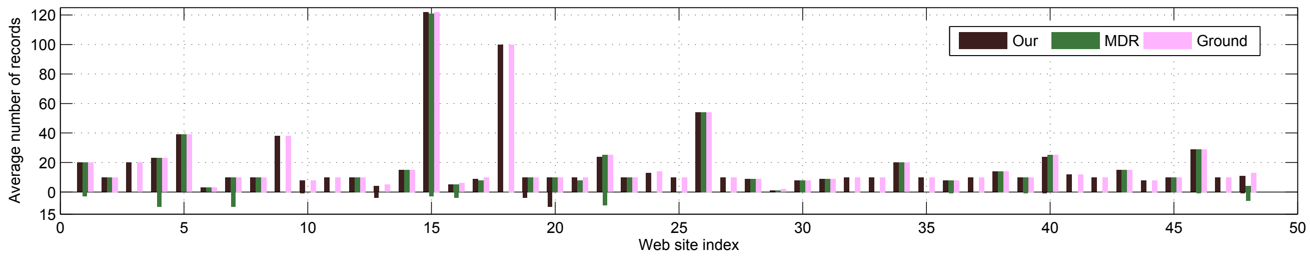


**Figure 5: Extracted records for each site in TB1, the reported number is the average over the number of pages in the site. TP number and FP number are shown by the bars above and below the axis respectively.**

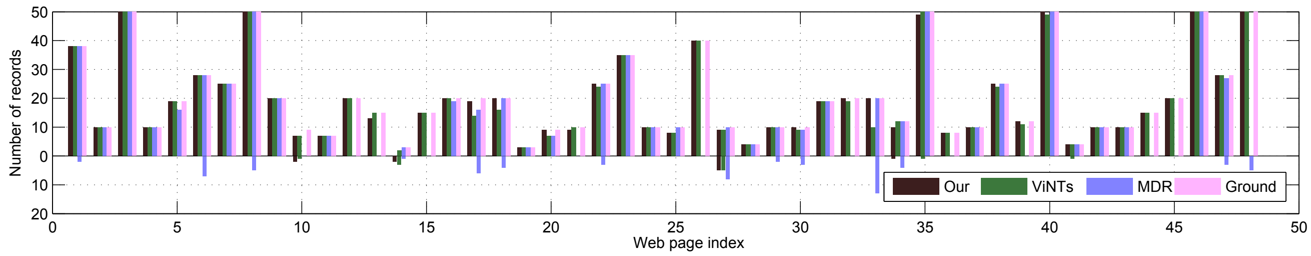


**Figure 6: Extracted records for each page in TB2. The legend is the same as that in Figure 5.**

On TB1, our method outperforms both MDR and TPC. The recall of our method is significantly better than that of MDR. In addition the precision of our method outperforms TPC about 7%. On TB2, the performance of our method is also much better than that of MDR. Compared with ViNTs, our method can extract more correct results, and achieve 2% improvement in R-mi. We can see that for MDR P-ma is much smaller than P-mi. It is because for some pages, although the MDR program terminated normally, it cannot give any output, thus both TP and FP are 0.

The details of the extracted records on TB1 and TB2 are given in Figure 5 and Figure 6 respectively. On TB1, our method outputs 10 false positives for site 20. After checking the pages manually, we find that each page in this site contains 10 recommended books on the right side bar. But the annotated ground truth only includes the books formatted with <table> in the center of the page. For this site, MDR only reports the books annotated. For site 35, we find that each field of a record is packed in a single subtree in **S**, such as id, title, URL, each of digest sentences, etc. Because different records have different number of subtrees, MDR fails to output any correct results. For site 48, each page contains more than one record regions, MDR only reports the largest one, and misses others. In Figure 6, we can see that for pages 17, 18 and 33, ViNTs misses some records. After checking these pages manually, we find that each of them has 3 or 4 regions. As reported by Zhao et al. [29], ViNTs is designed to extract records just from the major record region, it misses other smaller regions.

## 7.2 Nested and Intertwine Record Extraction

To collect data sets with nested and intertwine records, we investigated the online shopping Web sites one by one in an online shopping yellow page `http://www.toponlineshopping.com/`. There are 22 categories such as "Art & Collectibles" and "Beauty & Fragrances" in this page, and each category has about 6 sub-categories on average. Under each sub-category, we randomly selected 2 recommended Web sites. Instead of submitting queries to retrieve record pages, we directly clicked the navigation links in the home page, and obtained record pages. In this way, we successfully obtained record pages from 110 sites, and there are 50 sites adopting nested manner to present products, and 5 sites adopting intertwine manner. We downloaded 2 pages per nested site to construct the nested data set, thus it contains 100 Web pages. MDR terminated abnormally for one page in this data set. To construct the intertwine page set, we downloaded 3 pages per intertwine site, thus there are 15 pages in the intertwine data set.

**Table 4: Experimental results on nested pages.**

| | | Ground | TP | FP | P-mi | R-mi | P-ma | R-ma |
|---|---|---|---|---|---|---|---|---|
| ALL | Our | 1293 | 1292 | 2 | .998 | .999 | .997 | .998 |
| | MDR | 1293 | 1141 | 90 | .927 | .882 | .822 | .867 |
| ALL/wrong | Our | 1284 | 1283 | 2 | .998 | .999 | .997 | .998 |
| | MDR | 1284 | 1141 | 90 | .927 | .889 | .838 | .884 |

**Table 5: Experimental results on intertwine pages.**

| | Ground | TP | FP | P-mi | R-mi | P-ma | R-ma |
|---|---|---|---|---|---|---|---|
| Our | 262 | 259 | 0 | 1 | .989 | 1 | .986 |
| MDR | 262 | 253 | 50 | .835 | .966 | .898 | .967 |

The experimental results for nested and intertwine page sets are given in Tables 4 and 5. Our method achieves nearly perfect results, while MDR misses many records in nested record extraction, and gives more false positives in both nested and intertwine record extraction. After checking the pages for which MDR misses all records manually, we found that these pages format the records in a very complicated manner. In 3 pages, each <td> contains another <table> to encapsulate the information of one record. In another page, each <td> even contains two layers of nested <table>s. Our method can tackle these pages correctly since it does not rely on any particular tag. Our final record selection process effectively excludes noise records, while MDR outputs more false positives.

## 8. CONCLUSIONS

In this paper, we present a novel approach to extract data records in a Web page. The proposed RST structure can be utilized to address several key issues in the record extraction task. The two essential sub-tasks, namely, record region detection, and record segmentation, are handled in a unified manner with the proposed search pruning techniques on the RST structure. Different from the existing similarity-based methods, our method examines the similarity between the dynamically generated subtree groups taking into account the characteristics of the current record region. Owing to the pruning strategies, our method has a comparative complexity compared with the existing methods. Furthermore, we propose a new similarity measure in which each DOM node is regarded as an inseparable unit. Together with the detected tandem repeats, our similarity measure can tackle optional and repetitive fields in records properly. Extensive experiments are conducted to evaluate the performance of our method. The results demonstrate that our method can achieve very superior results for the test data sets, and different kinds of records can be tackled effectively.

## 9. REFERENCES

[1] `http://www.programmableweb.com/`.

[2] B. Adelberg. Nodose - a tool for semi-automatically extracting structured and semistructured data from text documents. *SIGMOD Rec.*, 27:283–294, June 1998.

[3] A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *SIGMOD '03*, pages 337–348, 2003.

[4] G. O. Arocena and A. O. Mendelzon. Weboql: restructuring documents, databases, and webs. *Theor. Pract. Object Syst.*, 5:127–141, August 1999.

[5] R. Baumgartner, G. Gottlob, and M. Herzog. Scalable web data extraction for online market intelligence. *Proc. VLDB Endow.*, 2:1512–1523, August 2009.

[6] D. Buttler, L. Liu, and C. Pu. A fully automated object extraction system for the world wide web. In *ICDCS '01*, pages 361–370, 2001.

[7] C.-H. Chang and S.-C. Lui. Iepad: information extraction based on pattern discovery. In *WWW '01*, pages 681–688, 2001.

[8] V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB '01*, pages 109–118, 2001.

[9] D. W. Embley, D. M. Campbell, Y. S. Jiang, S. W. Liddle, D. W. Lonsdale, Y.-K. Ng, and R. D. Smith. Conceptual model-based data extraction from multiple-record web pages. *Data Knowl. Eng.*, 31:227–251, November 1999.

[10] D. W. Embley, Y. Jiang, and Y.-K. Ng. Record-boundary discovery in web documents. In *SIGMOD '99*, pages 467–478, 1999.

[11] W. Gatterbauer, P. Bohunsky, M. Herzog, B. Krüpl, and B. Pollak. Towards domain-independent information extraction from web tables. In *WWW '07*, pages 71–80, 2007.

[12] D. Gusfield and J. Stoye. Linear time algorithms for finding and representing all the tandem repeats in a string. *J. Comput. Syst. Sci.*, 69:525–546, December 2004.

[13] A. Hogue and D. Karger. Thresher: automating the unwrapping of semantic content from the world wide web. In *WWW '05*, pages 86–95, 2005.

[14] C.-N. Hsu and M.-T. Dung. Generating finite-state transducers for semi-structured data extraction from the web. *Inf. Syst.*, 23:521–538, December 1998.

[15] N. Kushmerick. Wrapper induction: efficiency and expressiveness. *Artif. Intell.*, 118:15–68, April 2000.

[16] A. H. F. Laender, B. Ribeiro-Neto, and A. S. da Silva. Debye - date extraction by example. *Data Knowl. Eng.*, 40:121–154, February 2002.

[17] B. Liu, R. Grossman, and Y. Zhai. Mining data records in web pages. In *KDD '03*, pages 601–606, 2003.

[18] L. Liu, C. Pu, and W. Han. Xwrap: An xml-enabled wrapper construction system for web information sources. In *ICDE '00*, pages 611–621, 2000.

[19] W. Liu, X. Meng, and W. Meng. Vide: A vision-based approach for deep web data extraction. *IEEE Trans. on Knowl. and Data Eng.*, 22:447–460, March 2010.

[20] G. Miao, J. Tatemura, W.-P. Hsiung, A. Sawires, and L. E. Moser. Extracting data records from the web using tag path clustering. In *WWW '09*, pages 981–990, 2009.

[21] I. Muslea, S. Minton, and C. A. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems*, 4:93–114, March 2001.

[22] K. Simon and G. Lausen. Viper: augmenting automatic information extraction with visual perceptions. In *CIKM '05*, pages 381–388, 2005.

[23] W. Su, J. Wang, and F. H. Lochovsky. Ode: Ontology-assisted data extraction. *ACM Trans. Database Syst.*, 34:12:1–12:35, July 2009.

[24] J. Wang and F. H. Lochovsky. Data extraction and label assignment for web databases. In *WWW '03*, pages 187–196, 2003.

[25] Y. Yamada, N. Craswell, T. Nakatoh, and S. Hirokawa. Testbed for information extraction from deep web. In *WWW Alt. '04*, pages 346–347, 2004.

[26] Y. Zhai and B. Liu. Web data extraction based on partial tree alignment. In *WWW '05*, pages 76–85, 2005.

[27] Y. Zhai and B. Liu. Structured data extraction from the web based on partial tree alignment. *IEEE Trans. on Knowl. and Data Eng.*, 18:1614–1628, December 2006.

[28] Y. Zhai and B. Liu. Extracting web data using instance-based learning. *World Wide Web*, 10:113–132, June 2007.

[29] H. Zhao, W. Meng, Z. Wu, V. Raghavan, and C. Yu. Fully automatic wrapper generation for search engines. In *WWW '05*, pages 66–75, 2005.

[30] H. Zhao, W. Meng, and C. Yu. Mining templates from search result records of search engines. In *KDD '07*, pages 884–893, 2007.

[31] S. Zheng, R. Song, J.-R. Wen, and C. L. Giles. Efficient record-level wrapper induction. In *CIKM '09*, pages 47–56, 2009.

[32] S. Zheng, R. Song, J.-R. Wen, and D. Wu. Joint optimization of wrapper generation and template detection. In *KDD '07*, pages 894–902, 2007.

[33] J. Zhu, Z. Nie, J.-R. Wen, B. Zhang, and W.-Y. Ma. Simultaneous record detection and attribute labeling in web data extraction. In *KDD '06*, pages 494–503, 2006.