Web Record Extraction with Invariants

Zhijia Chen
Department of Computer and Information Sciences
Temple University
zhijia.chen@temple.edu

Weiyi Meng
Department of Computer Science
Binghamton University
meng@binghamton.edu

Eduard Dragut
Department of Computer and Information Sciences
Temple University
edragut@temple.edu

ABSTRACT

Web records are structured data on a Web page that embeds records retrieved from an underlying database according to some templates. Mining data records on the Web enables the integration of data from multiple Web sites for providing value-added services. Most existing works on Web record extraction make two key assumptions: (1) records are retrieved from databases with uniform schemas and (2) records are displayed in a linear structure on a Web page. These assumptions no longer hold on the modern Web. A Web page may present records of diverse entity types with different schemas and organize records hierarchically, in nested structures, to show richer relationships among records. In this paper, we revisit these assumptions and modify them to reflect Web pages on the modern Web. Based on the reformulated assumptions, we introduce the concept of invariant in Web data records and propose an algorithm to mine the invariants. Based on the mined invariants, we propose a bottom-up, recursive approach to construct the Web records. The proposed approach is both effective and efficient, outperforming the state-of-the-art Web record extraction methods by large margins on modern Web pages.

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/ZhijiaCHEN/VLDB-2023.

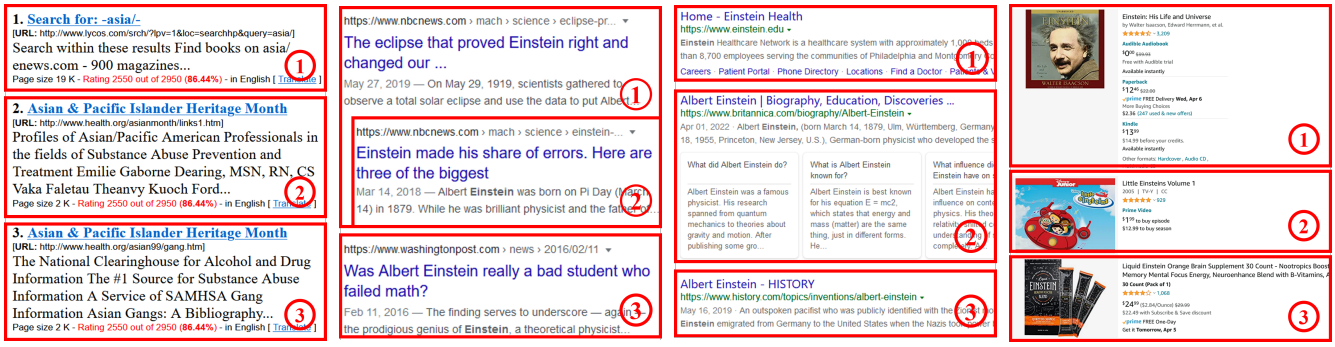
1 INTRODUCTION

Structured data is an important type of information on the Web. Such data is often in the form of records retrieved from underlying databases and displayed on Web pages according to some templates. We follow past literature and call them *data records*. Mining data records on the Web is useful because it enables one to integrate data from multiple Web sites and provide value-added services, e.g., comparative shopping and meta-querying. There is a rich literature on extracting data records from Web pages, dating back to the early days of the Web [14]. With very few exceptions, the approaches reported in the literature make either explicitly or implicitly two key assumptions: that records are drawn from a database with a uniform schema and that records are displayed in a linear structure on a Web page. Under such assumptions, data records extraction becomes a frequent pattern extraction exercise, where records are expected to be encoded in the HTML code using very similar HTML tags. That is commonly accompanied by some similarity measure on the HTML tag tree (or DOM tree) structure of a Web page, such as tag paths (a tag path of a node is the tag sequence from the root of the DOM tree to the node) [15, 28, 40], subtrees [2, 31, 43], and sibling nodes [33]. Measuring similarity is infeasible when records do not represent entities of the same kind and thus have different schemas. With the advent of Web 2.0, the structure variations of Web records have become much more significant resulting in an unpredictable performance for record extraction on the modern Web. As we will report in Section 5.4.1, the performance of several existing methods varies between 45% and 94% on modern Web pages.

To ease our ability to refer to older Web pages and compare them to modern ones in this work, we will refer to the traditional, older Web as Web 1.0 and the modern Web as Web 2.0. We observe two main factors that introduce unprecedented structure variations in Web 2.0 records: schema heterogeneity and the nonlinear (nested) structures. Schema heterogeneity refers to the situation where records in the same data region of a Web page have different numbers of data fields and data types. Search engines are good representatives of schema heterogeneity and the dramatic transformations they went through over the years. Twenty years ago, a Web search engine produced a list of results wrapped in a uniform structure in response to a query, providing a few metadata fields, including page title, page URL, and page snippets, as shown in Figure 1a. Nowadays, popular search engines such as Bing tailor the presentation of records to fit the content of the underlying Web pages, and they provide data fields according to the media type of its content. As an illustration, we show an example in Figure 1c, which shows 3 search records returned by Bing for the query "Einstein". Each record contains the traditional resource URL, page title, and content snippet, but their layouts and content structures are very different because Bing customizes their appearance and the displayed data fields according to their types. In this example, the first record is for a health company's main page, and it contains navigation links in the page; the second is for an encyclopedia page which contains some questions and their answers about Einstein; and the last is for a regular Web document. Modern e-commerce Web sites, like Amazon, exhibit similar behavior because they accommodate varied types of products, which are rendered differently, as illustrated by the search result records returned by Amazon for the query "Einstein" (Figure 1d); the returned products include a book, a TV episode, and food.

Nonlinear Web records are Web records with hierarchically nested structures. The nested structure of records was first introduced by social media posts via replies (a reply to a post becomes the child of that post) and then borrowed by Web sites to render

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX



(a) Web 1.0 records from the TBDW dataset [41].

(b) "Einstein" search results from Google.

(c) "Einstein" search results from Bing.

(d) "Einstein" search results from Amazon.

Figure 1: Examples of Web 1.0 and Web 2.0 records.

the list of records in a more compact way. For example, Google uses it to combine records retrieved from the same Web site, as shown in Figure 1b where record 1 and record 2 are from the same Web site, and the latter is nested under the former.

A few solutions emerged to address some of the above issues. For instance, MiBAT (Mining data records Based on Anchor Trees) [34] assumes that data records on a page must share some attributes. For example, one expects that all comments posted on a Web site to include a Posting Date. While this assumption is intuitive, such attributes are domain-specific, may be difficult to detect, and may not even exist. For example, some Web sites may label a comment's posting date as "just now," which is not a date. Another method is STEM [15], which uses a prefix tree of HTML tag path to reduce noise structures introduced by the complex data fields so the records are more regularly structured. However, the method only considers linear records, and the prefix tree cannot handle nested records. There are also solutions that are focused on records with user generated content, such as comments and postings, and thus they can leverage domain-specific features to help locate records and determine boundaries [6, 11, 19]. However, these are not generic record extraction solutions and do not give generic solutions.

In this paper, we first give the underlying assumptions that governed most of the literature on Web data record extraction to date, second, we show that they do not hold true in Web 2.0, third we argue that they need to be revisited (relaxed) and formulate new assumptions, and finally, we give a bottom-up extraction algorithm based on those assumptions. The proposed algorithm achieves high accuracy across multiple datasets (with F-scores between 0.93 and 0.95), both old and news proving empirically that it copes with the diverse record types and rendering methods of the modern Web.

Our solution is predicated on the presence of *invariants* in the way data records are presented on a Web page. The invariants may be part of the data records themselves (e.g., a common attribute) or the rendering HTML template. Our goal is to define such invariants, mine them and use them to reconstruct the data records.

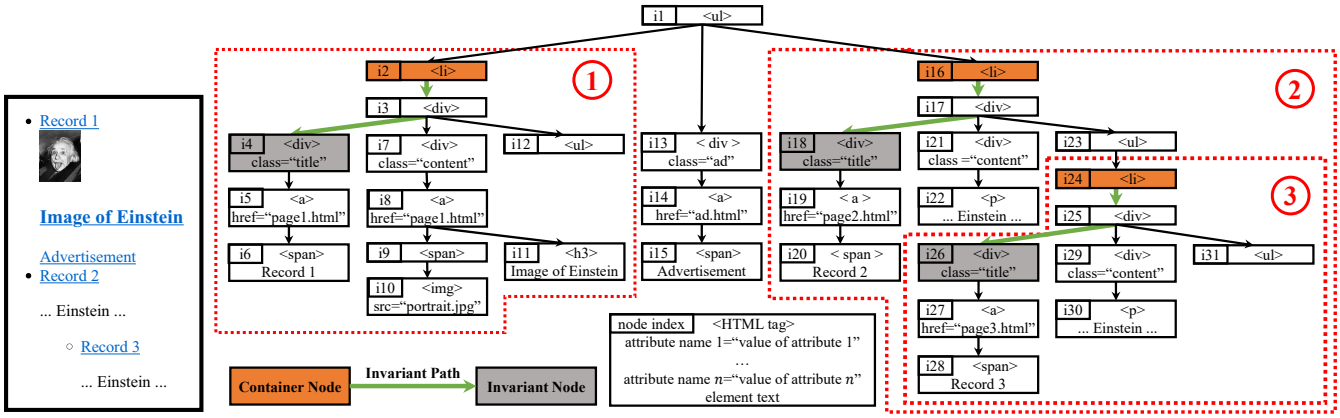
Here we give a high-level overview of our solution using the example in Figure 2. We first turn the DOM tree into a sequence of numbers via a pre-order traversal on the tree, where each node is mapped to a number using an encoding function. A node's index is its position in the sequence. We then mine frequent patterns from the sequence with signal processing techniques and try to map each frequent pattern into an invariant structure in the tree. For example, one possible invariant structure is present at the subtrees rooted at nodes $i4$, $i18$, and $i26$. We use those invariant structures as Web record anchors for locating data records by matching the paths from the anchors to the root of each record. Note that the paths from node $i4$ to $i2$, $i18$ to $i16$, and $i26$ to $i24$ are the same. We define two kinds of invariants: record level and path level.

We make the following contributions:

- We revise the assumptions followed by traditional Web record extraction tools and proposed new ones that are reflective of today's Web.
- We define invariants and propose novel algorithms using methods inspired by signal processing literature to mine such invariants from Web pages.
- We give a bottom-up approach for mining Web data records. To our knowledge, ours is the first such approach; all existing approaches are top-down.
- We conduct extensive experimental studies and show that our approach is both effective and efficient, outperforming the state-of-the-art (SOTA) Web record extraction methods.

2 RELATED WORK

The Web record extraction problem has been extensively studied, and various heuristic rules have been proposed. The existing approaches for Web record extraction can be classified by the level of automation: manual approaches, semi-automatic approaches, and automatic approaches. Manual approaches aim at writing specialized tools to extract Web records from a limited number of Web pages. They require manual inspection of each Web page and its HTML source code. Then one writes a specialized program in the form of overfitted rules (e.g., in the form of regular expressions) to target specific parts of the HTML source code. Manual approaches may achieve the most accurate results, but they are not scalable and require domain expertise to compose the rules. The semi-automatic approaches usually involve supervised or semi-supervised learning, which requires human-labeled pages as input to learn extraction rules and wrappers [7]. The automatic approaches focus on detecting frequent patterns based on the fact that records from the same list share very similar structure patterns both visually (as rendered on the target Web page) and in their underlying HTML



(a) Rendered Page **(b) DOM Tree**

Figure 2: Records with heterogeneous contents and nested structures. Record boundaries are marked with red dotted lines.

source codes. These approaches require certain similarity metrics to match potential records/patterns. They may be sensitive to the level of structural variation across records.

Automatic solutions target two extraction scenarios in general: extract records from a single Web page and extract one record per page with access to multiple pages from the same template [29]. This paper is focused on the former, and we present a number of relevant works here. There are HTML tags they are designed to format tables and lists in Web pages, such as `<table>`, `<ul>`, `<ol>`, etc. Thus, some works are dedicated to extracting records using these tags [4, 10, 13, 30, 38, 45]. These solutions do not generalize well across the Web, they tend to extract cleaner data and run much faster for Web pages that follow those specialized HTML tags, primarily pages that predate Web 2.0.

The DOM tree structure is arguably the most exploited modality by Web record extraction approaches. These works apply various pattern mining algorithms to locate record regions within a page by analyzing the DOM trees, such as repeating tags [3, 14], repeating tag paths [8, 15, 17, 21, 28, 36], and repeating subtrees [23, 43]. The three patterns inspire the three different node encoding schemes that we will discuss in Section 4.2. To extract each record from a record region, a key task is to determine the record boundaries. In an ideal case, records are flat and all have exactly the same structure, and the boundaries are determined by the repeating pattern. But in many cases, there are structure variations, and these methods generally define certain similarity measurements to compute the boundaries and align the records. Naturally, record alignment can be performed on the tree structure, with tag path alignment [15], sibling alignment [34], or tree alignment [24, 43, 44]. The similarity may also be performed on a sequential data structure, where the DOM tree is transformed into a sequence/string [1, 8, 20, 37] by traversing the tree in a certain order and assigning appropriate codes to nodes. Reducing the tree data structure to the sequence data structure introduces the opportunities of applying signal processing techniques to the Web record extraction problem, and our method follows this trend as well.

Apart from the DOM tree structure, some works leverage the visual layout in the rendered Web page [5, 16, 32]. There are also hybrid methods that combine the two [26, 46]. These methods tend to be less efficient because they are throttled by the page rendering process in tools like Selenium, which complicates the system design and reduces their robustness by introducing an extra CSS render engine into the process.

3 RECORD EXTRACTION REVISITED

We start the formal presentation of the problem in this section. We first discuss the underlying assumptions in mining Web data records. We then give the reasons those assumptions are violated in modern Web pages. Finally, we revise those assumptions to meet the design of modern Web pages. The discussion in this section assumes that the extraction algorithms use the HTML source code as the primary input for mining. This is generally true, but a number of algorithms also use visual features in rendered records in the browser in addition to the HTML source code [25, 26, 46]. Our discussion applies to them as well, but we do not explicitly discuss using visual features to keep our presentation focused.

3.1 Assumptions in Mining Web Data Records

The literature on mining Web data records makes three main assumptions (some explicitly, others implicitly) about the way data records are represented in the HTML code of a Web page:

Assumption 1. Web pages are organized in regions. The region that contains Web records is called a *data region*. A data region is "continuous" and presents data records about "similar" entities that have the same schema.

Assumption 2. Web records in the same data region are represented using "almost" the same sequence of HTML tags.

Assumption 3. A set of Web records in a data region are represented as child sub-trees of the "same" parent node in the HTML DOM tree.

We take each of these assumptions and explain with supporting examples why they are violated on the modern Web. The "continuous" condition in Assumption 1 was almost universally satisfied in Web 1.0. There were no social media platforms whose content needed to be integrated, and the ad industry had yet to understand the full potential of digital ad placement [27]. The common violation

of "continuous" is the (strategic) insertion of ads among records. The "similar" requirement was based on the observation that Web sites would return records about entities sharing the same schema in response to a query. This is violated nowadays because Web sites use increasingly sophisticated retrieval algorithms that go beyond the classic query-record similarity search and include additional dimensions of relevance, such as market basket analysis [39]. Thus, most Web sites today give records of diverse entity types in response to a query. Figure 1d shows some of the records returned by Amazon for the query "Einstein." The list includes books, a TV show, and food items. This is not unique to Amazon; Google, Bing, and DuckDuckGo also return a diverse set of entities in response to a query. For example, Google includes data records about companies, like Einstein Bros. Bagels and Einstein Healthcare Network, along with data records that describe Albert Einstein.

Informally, the "almost" condition in Assumption 2 is a consequence of the similarity condition in Assumption 1. If one assumes that the data records follow the same schema, then one expects their representations in the HTML source code to be identical modulo small variations, like the absence of one or two attributes (for instance the absence of Edition in a book record– It is usually due to the presence of a Null value in that attribute in the database.) This condition is clearly violated when the records are of different entity types. Violations may still occur even when the records denote entities of the same type. For instance, the records from Britannica and Wikipedia returned by Google in response to "Einstein" have different sets of attributes, Wikipedia has Siblings and Britannica has Subjects Of Study. The difference is even more dramatic between Einstein Bros. Bagels and Einstein Healthcare Network although both denote companies; the former has three attributes, while the latter has none. This is because Google is able to distinguish between their industry types, food versus health, and include attributes specific to each type.

Assumption 3 states that a set of similar Web records are formed by some child sub-trees of the same parent node. This was satisfied in Web 1.0 because a record was not expected to be a child of another record. This however changed in Web 2.0, where there may be hierarchical relationships between records. Hence, the descendant records of a record cannot all share a common parent node. Some Web sites use hierarchical relations to convey certain semantics. For example, in social media, a reply to a post becomes the child of that post and search engines like Google uses it to combine records on a topic retrieved from the same Web site, as shown in in Figure 1b where record 1 and record 2 are from the same Web site, and the latter is nested under the former. These differences are also fueled by the emergence of novel data management technologies. During Web 1.0, web databases were relational, hence one expected the records to inherit the schema of the underlying tables. Today many web databases are not relational [18]. For example, Google's Big Table [9] and Amazon's DynamoDB [12], are not relational and can better accommodate records with diverse schemas.

3.2 A Picture is Worth a Thousand Words

The discussion in the previous section gives a qualitative analysis of the differences between Web 1.0 and Web 2.0 records. One may nonetheless question if the evolution brought by Web 2.0 necessitates revisiting the record extraction algorithms. Perhaps, one may expect those violations to be occasional and the existing solutions to remain resilient after some small changes. In this section, we give a quantitative argument of the changes triggered by the violations of those assumptions.

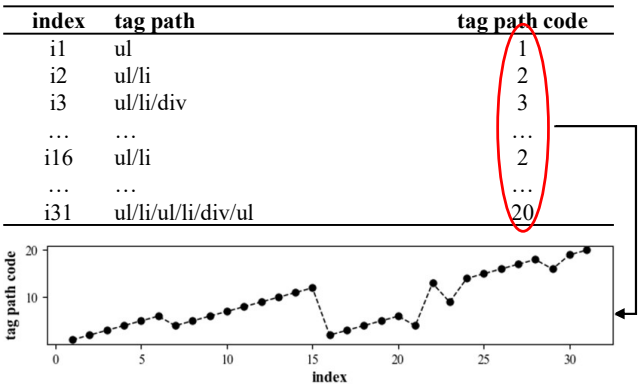


Figure 3: Tag path code sequence of D (Figure 2b).

We need a way to capture the observation that the presence of structured content in a Web page (aka data region) must exhibit an overall structural consistency, even in the presence of noise, so that it looks somewhat organized to a visual inspection. We inspire by visual data mining [22]. If the Assumptions 1 - 3 are consistently satisfied, we expect the records to be laid out so that it is obvious to a human's visual inspection that they are related to each other and, perhaps, referring to the same entity type. We draw inspiration from signal processing. Intuitively, suppose that we can map Web records to cycles in a signal, then if those assumptions are satisfied with high frequency we expect to see a signal with regular cycles (patterns). Otherwise, we should see little regularity in the cycles of the signal. We take the structure of the DOM tree of a Web page and flatten it into a sequence of integers, then we plot it into a graph. More specifically, each node is represented by its *tag path*, which is the string obtained by concatenating the HTML tags of the nodes on the path from the root node to the node in the DOM tree. For example, Figure 3 shows the tag paths of some nodes in D. We can assign a code for each unique tag path, then we get a sequence of tag path codes for the tree, as shown at the bottom of Figure 3. Details of the process can be found in [15, 28, 37].

Using tag path codes, we generate the sequential signal of data regions. We compare the sequential signals of data regions from Web 1.0 against those from Web 2.0. As representatives of Web 1.0 pages, we use samples from the TBDW dataset [41], which was a widely used benchmark dataset for Web record extraction. TBDW contains search results from 51 Web sites, collected in 2003 (see more details in Section 5.1). We choose three Web sites (1, 25, and 50) that are at the beginning, the middle, and the end of the dataset, and we randomly select a Web page from each of the three Web sites. As representatives of Web 2.0 records, we collected a large dataset of Web records from Google, Amazon, and news comments. We collected them in April 2022. We give the details in Section 5.1. We take at random a page from each of them. We do not plot the sequential signal of entire Web pages; we plot (zoom in) the signal of the main data region only.

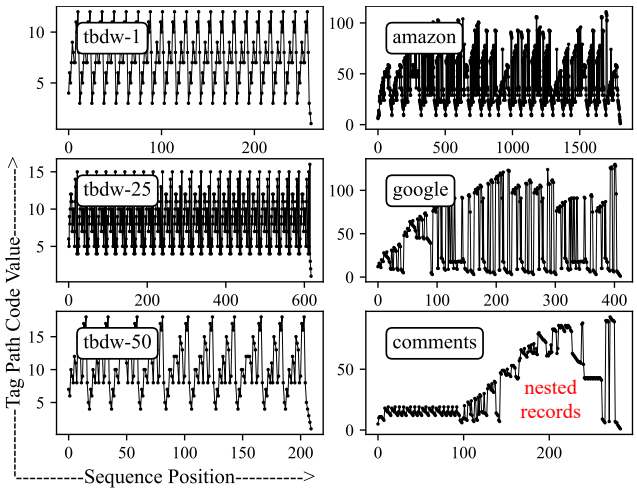


Figure 4: Signals of tag path codes of Web 1.0 (left column) and Web 2.0 records (right column).

Figure 4 gives the plots. The plots of the samples from the TBDW dataset are on the left and those from our dataset are on the right. One observes that the data regions of pages sampled from TBDW have neat repeating patterns, which indicates that there is little structure variation among records. Specifically, the sequential signal of tbdw-1 has a strict repeating pattern because there is no structure variation among its records; the sequential signals of tbdw-25 and tbdw-50 are not strictly periodic, but the irregular variations are minimal. In contrast, it is difficult to tell any periodical patterns in the sequential signals of the data regions of the pages sampled by us. Notice that most of the examples have some tag path codes repeated throughout their signals. We draw attention to the example from Google which contains an arch in its sequence. This corresponds to two nested records. Such behavior is not encountered in the data regions from the TBDW dataset. The apparent lack of cycles in the signal explains why all the baselines we tested on our dataset perform poorly, with F-scores between 0.45 and 0.66 (Section 5.4.1).

3.3 Assumptions Revisited

We showed in the previous section that the assumptions guiding the existing Web data record extraction algorithms no longer hold. Thus, they need to be revisited to reflect data regions in the modern Web. Specifically, Assumptions 1 - 3 need to be revised. For that, we need to distinguish between a *data record* and a *Web record*. A *Web record* has two parts: the data part, which is the *data record* retrieved from a database, and the *non-data* part, which includes control components (e.g., Add to Cart button) and HTML formatting components. For example, the data part of the second record in Figure 1d includes the cover image, "Little Einsteins Volume 1", "2005 | TV-Y | CC", "929", "$1.99", and "$12.99"; the non-data part includes the average rating stars (which will expand into the distribution of ratings when the mouse pointer is hovering on them) and all the invisible tags that help render the record.

Our premise when revising Assumptions 1 and 2 is that one can no longer expect data records as a whole to exhibit high similarity, but one nonetheless must expect Web records to share components that are similar. If one is able to identify such components, then we contend that one is able to reconstruct the Web records and extract the data with high accuracy. We call the presence of such components *record invariants*. They occur at multiple levels in Web records. The first level is the data part – even though data records may have different schemas in general, they may still share some attributes, like Product Price in an e-commerce Web site or the Posting Date of a user post on a social media platform. The second level is the non-data part. The non-data part usually follows a template that is not sensitive to the type of data record. For instance, records in many e-commerce Web sites have an Add to Cart button, and records from Google have an "About this result" button irrespective of their types. In addition, Web records tend to have consistent HTML formatting. For example, in Figure 1b, all the page titles are formatted with a <h3> tag along with an <a> tag. By including the non-data part as another source of record invariant mining, we are able to recover Web records even when they contain data records with very different schemas. Thus the similarity condition in Assumptions 1 and 2 becomes:

Assumption 4. Web records in a data region of a Web page contain one or multiple (non-trivial) record invariants where an invariant is a subtree structure that appears in every record.

Recall from the previous section that the presence of nested records in data regions are the main offenders of Assumption 3. The reason is that records in a data region form a tree of an arbitrary depth in the presence of nested records and, thus, Web records are not all children of the same parent. Consequently, we can no longer look for one node in the DOM tree to be the sole point of reference to guide record detection. Instead, we need to look for multiple such points of reference. Because Web records correspond to subtrees in a DOM tree, their root nodes are natural candidates. We introduce *path invariant*, which states that there is a certain record invariant such that the tag path from an *occurrence* of the invariant (i.e., a subtree) to its enclosing Web record's root node is the same among all the Web records in a data region. Hence, Assumption 3 becomes:

Assumption 5. Web records in a data region of a Web page contain one or multiple path invariants where a path invariant is a constant tag path between a Web record's root node to the occurrence of a certain record invariant.

To cope with the "continuos" requirement in Assumption 1 in the presence of breaks we assume that a data region is a collection of Web records sharing record and path invariants. This allows us to omit ad sections or other types of sections that are inserted between the records of a data region. With these assumptions, we propose a bottom-up approach to detect Web records in data regions. We can apply our approach recursively to identify all the data regions in a Web page. To our knowledge, all previous Web records extraction algorithms are top-down.

3.4 Notations

We model the DOM tree of an HTML document by an ordered tree (i.e., a directed acyclic graph with a designated root) and an ordering is specified for the children of each node. Given a DOM tree D, we assign each node an index using its depth-first traversal order starting from 1 and we refer to the ith node of D by D_i. We use $D(i)$ to refer to the subtree of D rooted at node i. Thus $D(1)$ is

the same as D. We denote a subtree relationship by $D(i) \subset D(j)$, with the meaning that the tree $D(i)$ rooted at node i is a subtree of $D(j)$ rooted at node j. For an ordered set A (such as an array or an ordered tree), we use A_i to refer to its ith element and $A[i:j]$ to refer to the sub-sequence of A that starts from A_i and ends at A_j.

We need to define record and path invariants as they occur in a DOM tree. A record invariant corresponds to a subtree in the DOM tree. We call the root of such a tree a *node invariant*. For example, the gray nodes in Figure 2b are node invariants, and the three subtrees rooted at nodes $i2$, $i18$, and $i26$ are three record invariants.

We need to introduce the concept of *container node* to define a path invariant. The container node of a record is the highest node in D such that all data fields of this record is under the node, and no data fields of any other records are under the node, except for the record's nested children. For example, nodes $i2$, $i18$, and $i26$ are the container nodes of the three records in Figure 2. Note that node $i18$ is the container node of record 2, and it only contains data fields of record 2 and record 3, and the latter is nested under the former.

With container nodes, the second type of invariant we observe is that the path from one record's container node to one of its invariant nodes (note that a record may have multiple invariant structures) is also invariant. Let r and s be two records in a data region in some DOM tree D, and n_r and n_s their container nodes, respectively. The tag paths in D from the root to n_r and n_s, respectively, may not be similar (e.g., when n_r is a nested record of n_s), but we observe that tag paths from their container nodes to their subtrees of a certain record invariant are indifferent to nested structures. We say such a path is a **path invariant**. For example, the green edges in Figure 2b are three path invariants for the three records, respectively. Note that even though record 3 is nested under record 2, the nested structure has no effect on the path invariant of record 3.

4 APPROACH

With Assumptions 4 and 5, we present **Miria** (**Mi**ne **r**ecord **i**nvari**a**nt), a bottom-up Web record extraction algorithm that detect records by first finding the record invariant among records and then matching their invariant paths to the corresponding record container nodes. In order to search for record invariant, we transform the DOM tree into a sequence. Each record invariant becomes a repeating pattern on the sequence. Thus, the invariant searching task becomes a sequential pattern mining problem. We provide an overview of our solution below and then present the detailed procedures step by step:

(1) Perform DFS on the DOM tree to transform it into an ordered node sequence.
(2) Assign a code to each node on the sequence.
(3) Mine frequent patterns on the code sequence.
(4) Map the frequent patterns back to the DOM tree to find the record invariant.
(5) Align Web records vertically by matching the invariant paths from the invariant nodes to their record container nodes.
(6) If there are gaps among sibling Web records, align them horizontally by matching their preceding and following subtrees.

4.1 Flatten The Tree

Following previous works [15, 28], we convert a DOM tree into a sequence to reveal the repeating structure patterns of Web records. Note that detecting a pattern requires the definition of equality between any pair of nodes on the sequence. So, we need to have a way to encode the nodes so that we can compare nodes based on their encoded values. We encode a node with a function f, and we write the encoding produced by f as $Encode_f(*)$, where the input may be a node or a tree (the output is a single code for the former and a code sequence for the latter).

Definition 4.1 (Node Encoding Sequence)**.** The Node Encoding Sequence (NES) of a tree D is a sequence of codes obtained by a node encoding function. In order to study the code sequence as a signal, we assign an integer to each unique node encoding in the sequence, and thus the NES becomes a sequence of positive integers. More specifically, during the NES constructing process, if we meet a node encoding for the first time, we assign the next unused integer (starting from 1) to the node encoding; otherwise, we use the integer that was assigned to the node encoding previously.

4.2 Choose Node Encoding Scheme

Our definition of NES is a generalization of the **Web Page Sequence (WPS)** [15] which is a sequence of nodes encoded by their HTML tag path, and we denote such encoding scheme by $Encode_{HTP}$.

Definition 4.2 (HTML Tag Path Encoding ($Encode_{HTP}$))**.** Given a DOM tree D, the $Encode_{HTP}$ of a node $x \in D$ is the tag path from the root of D to x.

For example, the $Encode_{HTP}$ for D_{i6} of Figure 2b is:

$$Encode_{HTP}(D_{i6}) =< \texttt{ul}, \texttt{li}, \texttt{div}, \texttt{div}, \texttt{a}, \texttt{span} > \quad (1)$$

Apparently, $Encode_{HTP}$ represents a node by its ancestor information. However, such an encoding scheme is not a good choice when there are nested structures. For example, in Figure 2b, we expect the nodes of the record invariant to have the same code, which cannot be satisfied if we encode a node using its tag path. This situation motivates the problem: how to encode a node so that the invariant nodes will have the same encoding while distinguishable from irrelevant nodes?

A naive solution is simply encoding a node by its tag. However, some tags are widely used in HTML documents, such as and , which can easily lead to false matching on an NES, i.e., matching with some patterns in the background noise, especially when the pattern is short. In order to reduce the chance of false matching, we enhance the tag of a node by including its attribute names, and we call it the *signature of a node*.

Definition 4.3 (Signature Encoding ($Encode_{SIG}$))**.** The $Encode_{SIG}$ of a node x is a tuple starting with the tag of x and followed by its assigned attribute names in alphabetical order.

For example, the $Encode_{SIG}$ of D_{i3} and D_{i4} are:

$$Encode_{SIG}(D_{i3}) =< \texttt{div} >, Encode_{SIG}(D_{i4}) =< \texttt{div}, \texttt{class} > \quad (2)$$

Apart from encoding a node by its signature, we also propose to encode a node by its structure. In fact, when we discuss similar

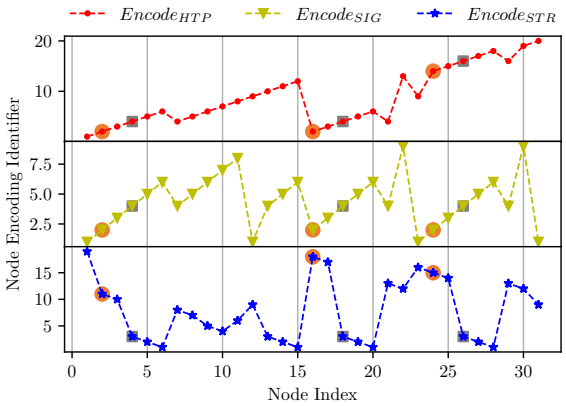


Figure 5: The NES of the example in Figure 2, encoded by $Encode_{HTP}$, $Encode_{SIG}$, and $Encode_{STR}$, respectively. The record container nodes are marked by orange circles, and invariant nodes are marked by gray squares.

structures of records, we have implicitly assumed that two nodes are equal if their subtree structures are the same.

Definition 4.4 (Structure Encoding ($Encode_{STR}$))**.** Given a DOM tree D, the $Encode_{STR}$ of a node $x \in D$ is a code that identifies $D(x)$'s structure, which is defined in a recursive way:

$$Encode_{STR}(x) := < Encode_{SIG}(x), Encode_{STR}(children(x)) > \quad (3)$$

where $Encode_{STR}(children(x))$ is the code array of x's direct children.

The $Encode_{STR}$ represents a node by its signature and its children's structure encoding (if any). We can find subtrees with the same structure by looking up their structure encoding identifier. In the trivial case where no structure variation exists, we can find Web records by building a histogram of structure encoding on a DOM tree with heuristic filters such as record size, text length, etc.

On choosing the node encoding scheme of the NES, we have two fundamental concerns: pattern recall and pattern precision. A higher pattern recall means more Web records being matched by a frequent pattern, while a higher pattern precision means less background noise being matched by a frequent pattern. We show the NES of D (Figure 2b) in Figure 5, constructed using $Encode_{SIG}$, $Encode_{HTP}$, and $Encode_{STR}$, respectively. The horizontal axis is the sequence index, while the vertical axis is the node encoding identifier. Each dot in a sequence stands for a node, and we place an orange circle on record container nodes and a gray square on invariant nodes. We see that for the NES using $Encode_{HTP}$, the third record does not match with any previous record because its tag path is deeper due to nesting. The NES using $Encode_{SIG}$ presents the most regular pattern, with all the 3 records matching with each other completely or partially. However, the subtrees of the page source information ($D(i4)$, $D(i18)$, and $D(i26)$), the snippet field of record 1 ($D(i7)$), and the advertisement ($D(i13)$), all share the same code sequence: <4,5,6>. At first glance, it seems that the NES with $Encode_{STR}$ shows a very slight sign of pattern signal. But one may notice how the sequence reflects the tree structure: nodes with the same code value have the same tree structure. By choosing a proper size filter – 3 in this case – we get <3,2,1> as the most frequent pattern, and it only matches with the subtree of the page source information in each record ($D(i4)$, $D(i18)$, and $D(i26)$) and the advertisement ($D(i13)$). The advertisement will be discarded later in the alignment process.

It is obvious that, if the target Web records share at least one record invariant (i.e., common subtree structure), then both $Encode_{SIG}$ and $Encode_{STR}$ can produce NES that has perfect pattern recall, whereas the NES encoded by $Encode_{HTP}$ will have perfect pattern recall only if the Web records are linear. The fundamental cause of their difference in pattern recall is that the code assigned by $Encode_{HTP}$ is dependent on the ancestors of a node.

Definition 4.5 (Constant Node Encoding)**.** A node encoding function f is a constant node encoding function if $Encode_f(D)$ is determined only by nodes in D.

Lemma 4.1. If Web records share at least one record invariant, then a constant node encoding function has perfect pattern recall.

A frequent pattern has higher discriminative power in matching records if the encoding function has less probability of encoding collision, i.e., the probability that a node x has the same encoding with a random node y, where x and y are from the same DOM tree D. Let N_{tag} and N_{att} denote the number of unique tags and the number of unique attribute names, respectively. Let d denote the depth of x and s denote the size of $D(x)$. For simplicity, we assume that the tags and attribute names have uniform distributions. Then we can estimate the probability of encoding collision for the three encoding schemes:

$$P(Encode_{SIG}) = \frac{1}{N_{tag}N_{att}} \quad (4)$$

$$P(Encode_{HTP}) = \frac{P(depth(x) = depth(y))}{N_{tag}^{d}} \quad (5)$$

$$P(Encode_{STR}) = \frac{P(layout(D(x)) = layout(D(y)))}{(N_{tag}N_{att})^{s}} \quad (6)$$

where $P(depth(x) = depth(y))$ is the probability that x and y have the same depth, and $P(layout(D(x)) = layout(D(y)))$ is the probability that there is an one-to-one top-down mapping between $D(x)$ and $D(y)$. Obviously, encoding collision is more likely to happen for $Encode_{SIG}$, while the chance for the other two decreases exponentially by the node depth or the corresponding subtree size. Moreover, the chance of layout equivalence is smaller than that of depth equivalence, thus structure encoding is expected to perform better in terms of pattern precision.

4.3 Construct NES

The NES of a DOM tree, with any of the node encoding schemes mentioned above, can be constructed in $O(N)$ time using depth-first traversal, where N is the size of the target DOM tree. As an example, we show the constructing process of NES using $Encode_{STR}$ in Algorithm 1. We first initialize the NES as an empty array and assign two empty maps for the node signature identifier and structure identifier, respectively (lines 1 - 3). Then we call the `dfs` procedure, inside which we first assign the two identifiers for each child by calling `dfs` recursively. Then we assign the node signature and structure encoding identifiers for the current node (lines 8 - 18), and we append the latter to the NES array.

Algorithm 1: Constructing NES using structure encoding.

```
Function NES_Using_Structure_Encoding(tree)
    NES ←empty array
    sigIdMap ←empty map
    strucIdMap ←empty map
    dfs(tree.root(), NES, sigIdMap, strucIdMap)
    return NES
Procedure dfs(node, NES, sigIdMap, strucIdMap)
    foreach child ∈ children of node do
        dfs(child, NES, sigIdMap, strucIdMap)
    end foreach
    sig ← node signature of node
    if sig ∉ sigIDMap then
        sigIDMap(sig) ← |sigIDMap|
    end if
    sigId ← sigIdMap(sig)
    struc ← empty array
    struc.append(sigId)
    foreach child ∈ children of node do
        struc.append(structure encoding ID of child)
    end foreach
    if struc ∉ strucIdMap then
        strucIdMap(struc) ← |strucIdMap|
    end if
    strucId ← strucIdMap(struc)
    NES.append(strucID)
```

4.4 Frequent Pattern Detection

4.4.1 Pattern Mining. With Lemma 4.1, we can theoretically locate every Web record by detecting frequent substrings (i.e., continuous subsequences) in an NES constructed by a constant encoding function. The frequent substring mining problem is a classical pattern mining problem and can be solved by using a suffix tree. The suffix tree is a compressed trie of all the suffixes of a given string and it enables fast implementations of many important string operations such as searching for repeated substrings or common substrings. We can build a suffix tree using the Ukkonen's algorithm with $O(N)$ time complexity [42]. For illustration, we built a suffix tree for the $Encode_{SIG}$ NES in the middle of Figure 5, and we show the first three branches of the suffix tree in Figure 6. In the figure, the red circle is the root, black circles are internal nodes and white circles are leaf nodes. Each node is labeled by a substring. Concatenating the node labels from the root to a leaf node produces a suffix of the original string. We can find the number of substrings with a given suffix by counting the number of leaf nodes under the internal node reached by the path starting with the suffix. For example, the figure shows that there are 4 substrings starting with $< 1 >$, and 3 substrings starting with $< 2, 3, 4, 5, 6, 4 >$. Note that the \$ symbol denotes the end of the string.

To mine frequent patterns, we need to set a threshold for the pattern frequency F_{th} and a threshold for the pattern length L_{th}. We first traverse the suffix tree and find internal nodes that: 1) the length of the substring concatenated from its path to the root is greater than L_{th}; 2) there are more than F_{th} leaf nodes under the internal node. If we set $F_{th} = 2$ and $L_{th} = 3$, we can get the following frequent patterns from Figure 6:

$< 1, 2, 3, 4, 5, 6, 4 >$, $< 2, 3, 4, 5, 6, 4 >$, $< 2, 3, 4, 5, 6, 4, 9, 1 >$, $< 3, 4, 5, 6, 4 >$, $< 3, 4, 5, 6, 4, 9, 1 >$.

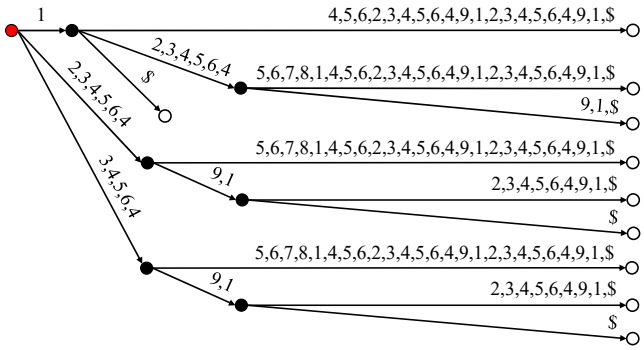


Figure 6: The first three branches of the suffix tree constructed from the $Encode_{SIG}$ NES of Figure 5

One quickly notices that there may be redundant patterns because some patterns are substring of other patterns. To remove redundancy, we will only keep closed patterns, as defined below.

Definition 4.6 (Super-pattern and Sub-pattern)**.** Given two patterns $p1$ and $p2$, where $p2$ is a substring of $p1$, we say $p1$ is a super-pattern of $p2$, and $p2$ is a sub-pattern of $p1$.

Definition 4.7 (Closed Pattern)**.** A pattern p is a closed pattern if any super-pattern of p has less support (i.e., frequency) than p.

A closed pattern is a minimal representation of its super-patterns without losing their support information [35]. In other words, we are still able to locate all the records with only closed patterns. Moreover, a closed pattern has more discriminative power than its sub-patterns in matching records, and the search space is significantly reduced by selecting closed patterns.

Thus, in the above patterns, we will keep the three closed patterns: $< 1, 2, 3, 4, 5, 6, 4 >$, $< 2, 3, 4, 5, 6, 4 >$, $< 2, 3, 4, 5, 6, 4, 9, 1 >$.

4.4.2 Pattern Reduction. So far, it is guaranteed that there is at least one pattern for Web records in the same data region. However, a pattern may span over multiple records, which makes it difficult, if not impossible, to decide the correct record container node in our following step. For example, the closed pattern $< 1, 2, 3, 4, 5, 6, 4 >$ has two occurrences in the $Encode_{SIG}$ NES of Figure 5, starting at indexes 1 and 23, respectively. Note that for each occurrence, the first node and the other five nodes do not belong to the same record (see the corresponding record boundaries in Figure 2b). In this step, our goal is to reduce the length of a candidate pattern such that **every occurrence of the pattern is within one and only one Web record**.

Given a pattern p mined from NES A that is derived from the DOM tree D, let O denote p's occurrences on A, and let $LCA(oi), oi \in O$ be a function that returns the lowest common ancestor of oi's corresponding nodes in D. We determine if we should reduce p's length by Lemma 4.2.

Lemma 4.2. If $oi, oj \in O, oi \neq oj$, and $LCA(oi) = LCA(oj)$, then o_i and o_j span over multiple records on A.

Proof. We prove the statement by contradiction. Suppose

Algorithm 2: Pattern reduction process.

input : A pattern p.
output : A reduced pattern.

```
1  p' ← ∅;
2  for i:=1 to length(p) step 1 do
3      for j:=length(p) to i step -1 do
4          O ← the occurrences of p[i : j];
5          if |{LCA(oi), oi ∈ O}| = |O| then
6              if length(p[i : j]) > length(p') then
7                  p' ← p[i : j];
8              end if
9              break;
10         end if
11     end for
12 end for
13 return p';
```

1) $LCA(oi) = LCA(oj)$, and

2) r_{oi} and r_{oj} are two different records containing oi and oj, respectively.

If $r_{oi} \not\subset r_{oj}$ and $r_{oj} \not\subset r_{oi}$, then obviously, $LCA(oi) \neq LCA(oj)$. Else, without loss of generality, assume that $r_{oi} \subset r_{oj}$, because oj does not span over multiple records, we have $oj \cap r_{oi} = \emptyset$, so $LCA(oj) \notin r_{oi}$. Since $oi \subset r_{oi}$, we have $LCA(oi) \in r_{oi}$, thus $LCA(oi) \neq LCA(oj)$. □

Note that Lemma 4.2 gives the sufficient but not necessary condition for patterns that span over multiple records. In practice, we found that the condition is very accurate in detecting such patterns. We show our process for pattern reduction in Algorithm 2, where we keep shrinking p until we meet the condition $|\{LCA(oi), oi \in O\}| = |O|$, and we return the largest sub-pattern p' that satisfy our condition. After pattern reduction, we apply the pattern length threshold one more time to evict trivial patterns produced by the reduction process.

Applying the Algorithm 2, the above three closed patterns are reduced to $< 2, 3, 4, 5, 6, 4 >$, $< 2, 3, 4, 5, 6, 4, 9, 1 >$. In the presence of multiple frequent patterns in one data region (patterns are from the same data region and their occurrences are interleaving), we will keep the one with the most support, and thus $< 2, 3, 4, 5, 6, 4 >$ is the final frequent pattern for this example.

4.5 Vertical Alignment

Our last step is to find record container nodes by each pattern candidate. We use a pattern's occurrences as anchors to locate and align Web records. We start the matching process from the anchor trees defined below.

Definition 4.8 (Anchor Tree)**.** Given a DOM tree D and its NES, for each occurrence o of a frequent pattern p derived from the NES, $D(LCA(o))$ is an anchor tree.

Take the $Encode_{STR}$ NES in Figure 5 as an example (blue dashed line with star markers in the bottom), the pattern $< 3, 2, 1 >$ has 4 occurrences starting at the indexes 4, 13, 18, and 26, respectively. By mapping the indexes back to D (Figure 2b), we see that the corresponding anchor trees are $D(i4)$, $D(i13)$, $D(i18)$, and $D(i26)$.

Algorithm 3: Align Web records vertically by matching the paths from anchor trees to record container nodes.

input : a set of anchor trees A
output : a set of record container nodes

```
1  C ← A;
2  while |C| > 0 do
3      P ← ∅;
4      foreach c ∈ C do
5          p ← the parent node of c;
6          if ∀y ∈ C, y ⊂ p then
7              return C;
8          end if
9          add p to P;
10     end foreach
11     G ← P divided into groups such that the nodes in the
         same group have the same signature;
12     C ← the largest group of G;
13 end while
14 return C;
```

Please note that in this example the LCA node happens to be the first node in each sub-sequence.

An anchor tree inside a Web record is either a record invariant or some background noise that happens to have the same structure as a record invariant. By Assumption 5, we can find the record container nodes by traversing the ancestors of anchor trees and matching their invariant paths.

We show the process in Algorithm 3. Starting from a group of anchor trees, we initialize the record container nodes with the root nodes of the anchor trees. Inside the loop, we find the parent node for each container node and check if we have reached the lowest common ancestor of current container nodes, which means we have reached the boundary node of the record group and should stop (lines 5 - 8). Otherwise, we collect the parent nodes and regroup them by their node signatures (lines 9 - 11). Because an anchor tree is derived from a frequent sub-sequence that may not belong to the target records, when we group the parent nodes, we will have multiple groups if there are different parent node signatures (line 11). Then we update the container nodes with the largest group and go into the next iteration (line 12).

Take the DOM tree in Figure 2 as an example. If we use the frequent pattern $< 3, 2, 1 >$ from the $Encode_{STR}$ NES, then we will get the group of anchor trees rooted at $< D_{i4}, D_{i13}, D_{i18}, D_{i26} >$, which are used to initialize the record container nodes. In the first iteration, the container nodes are updated to $< D_{i3}, D_{i1}, D_{i17}, D_{i25} >$. They will be split into two groups: $< D_{i3}, D_{i17}, D_{i25} >$ and $< D_{i1} >$. Because D_{i1} has a different node signature from the others, we select the larger group. In the next iteration, the group is updated to $< D_{i3}, D_{i17}, D_{i25} >$. In the next iteration, the parent node of D_{i2}, i.e., D_{i1}, is the lowest common ancestor of all the container nodes, and we stop the loop. We return the final record container nodes $< D_{i2}, D_{i16}, D_{i24} >$.

Algorithm 4: Align Web records horizontally by matching the preceding sibling nodes of record container nodes.

input : a set of record container nodes C
output : a set records where each record is a list of subtrees

```
1  R ← {[c], c ∈ C};
2  while |R| > 0 do
3      foreach r ∈ R do
4          p ← the preceding node of r_0;
5          if p = null ∨ p ∈ C then
6              foreach r ∈ R do
7                  append r's following nodes that are
                     unclaimed to r;
8              end foreach
9              return R;
10         end if
11         insert p to the top of r;
12     end foreach
13     G ← R divided into groups such that the preceding
         nodes of nodes in the same group have the same
         signature;
14     R ← the largest group of G;
15 end while
16 return R;
```

4.6 Horizontal Alignment

So far we have assumed that a Web record is embedded in a single subtree. This is a safe assumption for Web 2.0 records (the authors never encountered a violation on the modern Web). However, in the Web 1.0 era, a record may be presented by multiple continuous subtrees. To handle this case, we add an optional step of horizontal alignment to include the preceding and following subtrees for records detected in the last step.

Similar to the vertical alignment process, we align Web records horizontally by matching their preceding sibling nodes, and we show the process in Algorithm 4. We first construct an array from each container node (line 1) and then we keep appending the preceding node to each array until there is no preceding node available or the preceding node is also a container node (lines 4 - 11). Finally, for each array, we append the following sibling nodes that are not claimed by any other array.

4.7 Time Complexity

Suppose the target DOM tree has N nodes, and the height of the tree is H. Let N_p be the number of patterns, N_o be the average number of pattern occurrences, and L_p be the average length of a pattern. The time complexity of the NES construction step and the frequent pattern mining step are both $O(N)$. The pattern reduction step has a quadratic running time w.r.t. L_p, which is generally smaller than N by several orders and thus negligible. At the record matching step, for each pattern, there are at most H matching iterations, and each iteration takes $O(N_o)$ time. Thus the running time in this step is $H \cdot N_p \cdot N_o$. Note that $N_p \cdot N_o < N$, thus $H \cdot N_p \cdot N_o$ is bounded by $O(HN)$. Overall, the running time of the solution has a loose up-bound of $O(HN)$. In practice, DOM trees tend to grow horizontally rather than vertically for data regions. For example, the DOM trees of result Web pages of a search engine have very similar tree heights irrespective of the number of records in each Web page. In the four datasets used in this paper (see Table 1), their average tree heights are similar (range from 10 to 20) while their average tree sizes are dramatically different (range from 400 to 5000). Hence, if we regard H to be a constant or a small integer, our solution has an almost linear running time in N in practice. This is confirmed by the efficiency experiments in Section 5.5.

5 EXPERIMENTS

Our evaluation centers around three questions: (1) does our method outperform the SOTA? (2) does our method cope with record heterogeneity well? And (3) is the efficiency of our method competitive compared to that of the baselines?

We implement our method in Python, and we compare it with several baselines (see below). We compare their effectiveness using precision and recall. We evaluate them on multiple datasets, which we believe are good representatives of Web pages from Web 1.0 and Web 2.0. For efficiency, we compare the running time of our method with those of the baselines that are also implemented in Python. All the experiments are performed on a PC with an Intel CPU@3.6 GHz and 32 GB RAM@2666 MHz.

5.1 Datasets

We conduct our experiments on the following datasets. As a representative of Web 1.0, we use the **TBDW** dataset [41]. It includes 114,540 results pages from 51 deep Web sites. It was created in 2003 and still serves as a benchmark. We argue that a new benchmark needs to be created reflective of today's Web. According to http://httparchive.org/, the average Web page size has increased by at least 363% from 2010 to March 2022. For this study, we create more challenging datasets that represent i) the latest Web programming paradigms and ii) Web pages with nested regions. We constructed 3 datasets in April 2022. The **AMAZON** dataset consists of product search results. We use the top 100 Amazon searches of the year 2021 in the U.S. (www.semrush.com/blog/most-searched-items-amazon), and we keep the first result page for each query. To create the **GOOGLE** dataset, we use the top 300 queries of the year 2021 in the U.S. (trends.google.com/trends/yis/2021/US); we keep the first two result pages for each query. Google's first page is a good representative of modern Web pages with complex structures, including Twitter feeds, Google Map and News, which are interleaved between the classic list of results. It also includes nested records. The second page is not as complex, but it still includes many hard cases. The **COMMENT** dataset consists of comments posted at news outlets. It has 2,000 comment sections, each having at least 10 comments, uniformly distributed over 100 Web sites.

For all the datasets, we manually create the XPATHs expressions to label the record container elements in each Web site. We present some basic statistics of the datasets in Table 1.

5.2 Baselines

We implemented our algorithm with each of the three node encoding schemes introduced in Section 4.2. We compare our method with three baselines: (1) **DEPTA** [43] is a widely referenced baseline in the literature; (2) the method proposed by **Velloso et al.**

Table 1: Statistics of the datasets.

dataset	# Web sites	# pages	# records	avg. pages/Web site (std.)	avg. records/page (std.)	avg. size (std.)	avg. height (std.)
TBDW	51	255	2647	5 (0)	10.38 (17.93)	406.01 (293.89)	12.84 (14.20)
AMAZON	1	100	4834	100 (0)	48.34 (19.33)	5530.23 (1471.07)	20.05 (6.67)
GOOGLE	1	300	3155	300 (0)	10.52 (3.17)	1492.78 (801.16)	19.72 (9.567)
COMMENT	100	2000	60259	20 (0)	30.13 (27.58)	1088.61 (994.34)	13.36 (5.83)

[37] was chosen as a representative of the SOTA; (3) **MiBAT** [33] is chosen because it includes the notion of domain-specific invariant patterns. **MiBAT** looks for invariants in data records (e.g., common attributes), and it expects the invariants to be manually defined. This requirement reduces our ability to test it against all datasets. For AMAZON, we use the pattern of "$" followed by a number to define the attribute Price as an invariant. For GOOGLE, we define a pattern with the tags <a>, <h3> and <cite>. For COMMENT, we use the attribute Post Date as the invariant. We do not evaluate MiBAT on TBDW since the dataset is collected from multiple domains, and we are not able to define an invariant pattern that works for all of them. In addition, we implement MiBAT based on our own understanding of the algorithm as described in the paper.

5.3 Evaluation Metrics

5.3.1 Record Recall and Precision. We measure a method's effectiveness by record recall and precision. However, although our label and prediction for a record are both a node, it is problematic to compute recall and precision by simply comparing the detected nodes with the labeled nodes. First, from the perspective of recall, our method locates a record by its record container node, which is defined to be the highest node that contains all the data fields of the record. But in practice, a node should be considered a correct container as long as it covers all data fields of one and only one record except for its nested records. For example, in Figure 2b, D_{i2}, D_{i16}, and D_{i24} are the record container nodes for the three records, but D_{i3}, D_{i17}, and D_{i23} should be considered correct record container nodes as well. Second, from the perspective of precision, a generic record extraction algorithm may detect other structured data apart from labeled records, which should be excluded from the accuracy measurement as long as they are not mistaken to be from the same group as the targetted records. For example, if there are multiple ads inserted among targetted Web records, and a method detects a group of the targetted Web records and a group of ads, the ads should not be counted as false positives if the method can correctly distinguish the two types of records. On the other hand, if the method only produces one record group that contains both the targetted records and the ads, then the ads should be considered mistakes. Moreover, the DEPTA and the method of Velloso et al. output the text contents of the detected records other than the container nodes, and it is difficult, if not impossible, to know from which node a record's contents are extracted. So, we match two records by their context contents and measure the recall and precision with the following strategy.

Let R denote the set of labeled records, and $\hat{G}$ denote the set of the predicted record groups where each group in $\hat{G}$ is a set of records discovered by the same pattern (ours and MiBAT) or a set of records from the same data region (DEPTA and Velloso et al.). We measure the accuracy on a subset of predicted records $\hat{R}$ which is the union of some predicted record groups in $\hat{G}$. A predicted group $\hat{g} \in \hat{G}$ is selected only if there exists a pair of records $r \in R, \hat{r} \in \hat{g}$ such that $text(r) = text(\hat{r})$. That is, we select a predicted record group only if it matches at least one labeled record by the text content. Then the set of hit records (i.e., true positives) is defined as:

$$hit = \{text(r), r \in R\} \cap \{text(\hat{r}), \hat{r} \in \hat{R}\} \tag{7}$$

We compute record recall and precision of $\hat{G}$ w.r.t R as:

$$\text{record recall} = \frac{|hit|}{|R|}; \text{record precision} = \frac{|hit|}{|\hat{R}|} \tag{8}$$

5.3.2 Anchor Recall and Precision. Anchor trees play a key role in Miria, and the discovery of an anchor tree is the prerequisite for detecting a Web record. We want to choose a node encoding scheme that has enough generalization power to detect every targetted Web record while avoiding producing invalid anchor trees that may introduce false positives to our results by biasing our record alignment process. Here we distinguish between the true and false anchor trees: a true anchor tree is an anchor tree that belongs to a Web record, and a false one does not belong to any targetted Web record. For example, using the frequent pattern <3,2,1> from $Encode_{STR}(D)$ (Figure 5), we get 4 anchor trees in D: $D(i2)$, $D(i16)$, $D(i24)$, and $D(i13)$. Obviously, only the first three anchor trees are true anchor trees. Thus, given a group of records R, let A be a set of anchor trees generated from a pattern that is shared by R, we define the anchor recall and anchor precision of A by:

$$\text{anchor recall} = \frac{\text{number of true anchor trees in } A}{|R|} \tag{9}$$

$$\text{anchor precision} = \frac{\text{number of true anchor trees in } A}{|A|} \tag{10}$$

5.4 Effectiveness Analysis

5.4.1 Record Accuracy. We compare our method with the baselines on the datasets, and Table 2 shows the average record recall and precision for each dataset. On the Web 1.0 dataset TBDW, both our method with $Encode_{STR}$ encoding scheme and DETPA achieved the best F1 score at 0.94. When it comes to Web 2.0 datasets, our methods always achieved the best F1 score, with the $Encode_{STR}$ winning on the GOOGLE and COMMENT datasets and the $Encode_{HTP}$ winning on the AMAZON dataset. It is noteworthy that the $Encode_{HTP}$'s recall is perfect on the AMAZON dataset but relatively poor on the GOOGLE and COMMENT datasets. This is because the $Encode_{HTP}$ cannot handle nested records, as we have discussed previously, and the AMAZON dataset is the only Web 2.0 dataset that does not contain nested records. The baselines all suffer great recall losses on the Web 2.0 records, except that MiBAT achieved a recall of 0.95 on the AMAZON dataset, which can be explained by the non-nested structure of this dataset as well.

Table 2: Comparison of recall, precision, and F1 score of different methods on the datasets.

method		TBDW			AMAZON			GOOGLE			COMMENT		
		Recall	Precision	F1	Recall	Precision	F1	Recall	Precision	F1	Recall	Precision	F1
Miria (Ours)	$Encode_{SIG}$	0.92	0.87	0.89	0.99	0.73	0.84	**1.00**	0.12	0.21	**0.97**	0.70	0.81
	$Encode_{HTP}$	0.91	0.86	0.89	**1.00**	0.94	**0.96**	0.85	0.46	0.60	0.68	**0.99**	0.80
	$Encode_{STR}$	**0.96**	0.92	**0.94**	0.95	0.95	0.95	0.93	0.93	**0.93**	0.96	0.95	**0.95**
DEPTA [43]		0.89	**0.99**	**0.94**	0.61	**0.98**	0.75	0.30	0.91	0.45	0.42	0.94	0.58
Velloso et al. [37]		0.94	0.92	0.93	0.80	0.94	0.87	0.41	0.90	0.56	0.47	0.90	0.61
MiBAT [33]		-	-	-	0.95	0.93	0.94	0.49	**1.00**	0.66	0.66	0.99	0.79

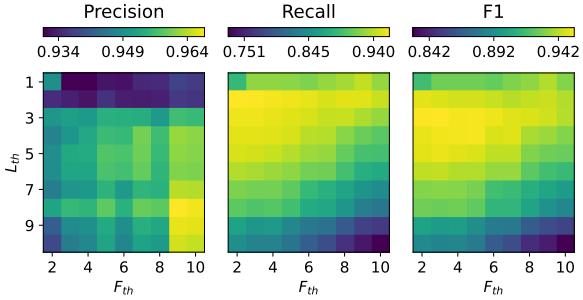


Figure 7: Sensitivity analysis of the frequency threshold and pattern length threshold.

5.4.2 Anchor Accuracy. We measure the anchor tree recall and precision of the three encoding methods on COMMENT and show the results in Table 3. We set $L_{th} = 3$, and if the labeled records of a page share multiple frequent patterns, we choose the one with the most support. We see that the $Encode_{STR}$ has the best precision while the $Encode_{SIG}$ has the worst, which can be explained by the node encoding collision probability given by Equations 4, 5, and 6. As for the recall, both the $Encode_{SIG}$ and $Encode_{STR}$ achieved a near-perfect score of 0.98, and the marginal loss is due to the pattern length threshold in producing anchor trees. The anchor trees produced by $Encode_{HTP}$ only covered 85% of the records due to the HTP variance in nested records.

Table 3: Anchor precision and recall.

	$Encode_{SIG}$	$Encode_{HTP}$	$Encode_{STR}$
precision	0.90	0.96	0.97
recall	0.98	0.85	0.98

5.4.3 Sensitivity Analysis. We perform the sensitivity analysis of our method with $Encode_{STR}$ to study how the pattern length threshold and the frequency threshold affect the performance. We test the method on the COMMENT dataset with L_{th} varying from 1 to 10 and F_{th} varying from 2 to 10. We show the results as a heat map in Figure 7. It is clear that the precision has a strong positive correlation with L_{th} and F_{th} while the recall has a negative correlation, which conforms to our analysis of the two parameters: longer pattern and higher frequency help to filter out noise, but impose a higher risk of missing records. However, we notice that the precision is much less sensitive to the two parameters as the score is ranging between 0.930 and 0.968, whereas the recall is ranging between 0.727 and 0.963. Overall, the best F1 score is 0.954 with L_{th} and F_{th} both equal to 3.

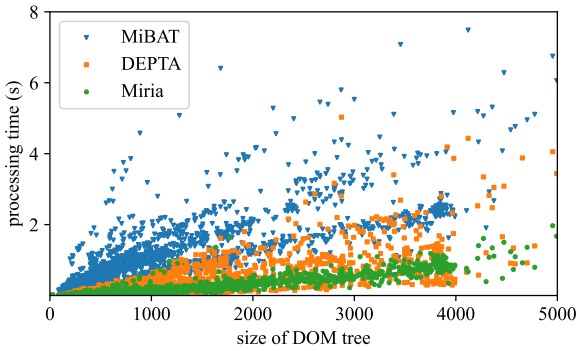


Figure 8: Distribution of processing time vs DOM tree size.

5.5 Efficiency Analysis

We compare the efficiency of Miria (with $Encode_{STR}$) against MiBAT and DEPTA. For fairness, all the three methods are implemented in Python. We test the methods on COMMENT and show the distribution of the processing time against the size of the DOM tree for the three methods in Figure 8. Overall, our method has the best running time, and it shows a stable linear increasing trend as the size of the DOM tree increases. The running time of the other two methods has greater variances, and MiBAT is the slowest due to the heavy overhead in recognizing posting-date strings.

6 CONCLUSION

In this paper, we revisited the Web record extraction problem on the modern Web, where Web records may present significant structure variations due to heterogeneous contents and nested structures. Existing solutions heavily rely on structure similarity and thus cannot handle the new challenges well. We proposed an effective and efficient solution that locates Web records by mining the invariant structures among records and then growing the invariant structures into records. We transformed the DOM tree representation of a Web page into a sequence representation, converting the problem of identifying invariant structures a frequent sequence pattern mining problem. We analyzed various node encoding schemes to map a tree node to a sequence code, both analytically and empirically. Our experiment results on multiple datasets showed that compared to the baselines, our proposed method with the $Encode_{STR}$ node encoding scheme has great advantages in extracting modern Web records while remaining competitive in extracting Web 1.0 records. We also showed that our method has linear running time in practice. Given the diversity of data and presentation variations on the Web today, there is a need for renewed focus on designing novel Web record extraction methods.

REFERENCES

[1] Arvind Arasu and Hector Garcia-Molina. 2003. Extracting structured data from web pages. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. 337–348.

[2] Lidong Bing, Wai Lam, and Yuan Gu. 2011. Towards a unified solution: data record region detection and segmentation. In *Proceedings of the 20th ACM international conference on Information and knowledge management*. 1265–1274.

[3] David Buttler, Ling Liu, and Calton Pu. 2001. A fully automated object extraction system for the World Wide Web. In *Proceedings 21st International Conference on Distributed Computing Systems*. IEEE, 361–370.

[4] Michael J Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. 2008. Webtables: exploring the power of tables on the web. *Proceedings of the VLDB Endowment* 1, 1 (2008), 538–549.

[5] Deng Cai, Shipeng Yu, Ji-Rong Wen, and Wei-Ying Ma. 2003. Extracting content structure for web pages based on visual representation. In *Asia-Pacific Web Conference*. Springer, 406–417.

[6] Donglin Cao, Xiangwen Liao, Hongbo Xu, and Shuo Bai. 2008. Blog post and comment extraction using information quantity of web format. In *Asia Information Retrieval Symposium*. Springer, 298–309.

[7] Chia-Hui Chang, Mohammed Kayed, Moheb R Girgis, and Khaled F Shaalan. 2006. A survey of web information extraction systems. *IEEE transactions on knowledge and data engineering* 18, 10 (2006), 1411–1428.

[8] Chia-Hui Chang and Shao-Chen Lui. 2001. IEPAD: Information extraction based on pattern discovery. In *Proceedings of the 10th international conference on World Wide Web*. 681–688.

[9] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. 2008. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)* 26, 2 (2008), 1–26.

[10] Xu Chu, Yeye He, Kaushik Chakrabarti, and Kris Ganjam. 2015. Tegra: Table extraction by global record alignment. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*. 1713–1728.

[11] Fan Chun-Long and Meng Hui. 2012. Extraction technology of blog comments based on functional semantic units. In *2012 IEEE International Conference on Computer Science and Automation Engineering (CSAE)*, Vol. 3. IEEE, 422–426.

[12] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. Dynamo: Amazon's highly available key-value store. *ACM SIGOPS operating systems review* 41, 6 (2007), 205–220.

[13] Hazem Elmeleegy, Jayant Madhavan, and Alon Halevy. 2009. Harvesting relational tables from lists on the web. *Proceedings of the VLDB Endowment* 2, 1 (2009), 1078–1089.

[14] David W Embley, Yuan Jiang, and Y-K Ng. 1999. Record-boundary discovery in Web documents. In *Proceedings of the 1999 ACM SIGMOD international conference on Management of data*. 467–478.

[15] Yixiang Fang, Xiaoqin Xie, Xiaofeng Zhang, Reynold Cheng, and Zhiqiang Zhang. 2018. STEM: a suffix tree-based method for web data records extraction. *Knowledge and Information Systems* 55, 2 (2018), 305–331.

[16] Wolfgang Gatterbauer, Paul Bohunsky, Marcus Herzog, Bernhard Krüpl, and Bernhard Pollak. 2007. Towards domain-independent information extraction from web tables. In *Proceedings of the 16th international conference on World Wide Web*. 71–80.

[17] Tomas Grigalis. 2013. Towards web-scale structured web data extraction. In *Proceedings of the sixth ACM international conference on Web search and data mining*. 753–758.

[18] Jing Han, Ee Haihong, Guan Le, and Jian Du. 2011. Survey on NoSQL database. In *2011 6th international conference on pervasive computing and applications*. IEEE, 363–366.

[19] Huan-An Kao and Hsin-Hsi Chen. 2010. Comment Extraction from Blog Posts and Its Applications to Opinion Mining.. In *LREC*. Citeseer, 1113–1120.

[20] Mohammed Kayed. 2012. Peer matrix alignment: a new algorithm. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 268–279.

[21] Mohammed Kayed and Chia-Hui Chang. 2009. FiVaTech: Page-level web data extraction from template pages. *IEEE transactions on knowledge and data engineering* 22, 2 (2009), 249–263.

[22] Daniel A Keim. 2002. Information visualization and visual data mining. *IEEE transactions on Visualization and Computer Graphics* 8, 1 (2002), 1–8.

[23] Bing Liu, Robert Grossman, and Yanhong Zhai. 2003. Mining data records in web pages. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. 601–606.

[24] Bing Liu and Yanhong Zhai. 2005. NET–a system for extracting web data from flat and nested data records. In *International Conference on Web Information Systems Engineering*. Springer, 487–495.

[25] Wei Liu, Xiaofeng Meng, and Weiyi Meng. 2006. Vision-based Web Data Records Extraction.. In *WebDB*.

[26] Wei Liu, Xiaofeng Meng, and Weiyi Meng. 2009. Vide: A vision-based approach for deep web data extraction. *IEEE Transactions on Knowledge and Data Engineering* 22, 3 (2009), 447–460.

[27] Andrew John McStay. 2016. *Digital advertising*. Macmillan International Higher Education.

[28] Gengxin Miao, Junichi Tatemura, Wang-Pin Hsiung, Arsany Sawires, and Louise E Moser. 2009. Extracting data records from the web using tag path clustering. In *Proceedings of the 18th international conference on World wide web*. 981–990.

[29] Adi Omari, Benny Kimelfeld, Eran Yahav, and Sharon Shoham. 2016. Lossless separation of web pages into layout code and data. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 1805–1814.

[30] Disheng Qiu, Luciano Barbosa, Xin Luna Dong, Yanyan Shen, and Divesh Srivastava. 2015. Dexter: large-scale discovery and extraction of product specifications on the web. *Proceedings of the VLDB Endowment* 8, 13 (2015), 2194–2205.

[31] Shengsheng Shi, Chengfei Liu, Chunfeng Yuan, and Yihua Huang. 2014. Multi-feature and dag-based multi-tree matching algorithm for automatic web data mining. In *2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, Vol. 1. IEEE, 118–125.

[32] Kai Simon and Georg Lausen. 2005. ViPER: augmenting automatic information extraction with visual perceptions. In *Proceedings of the 14th ACM international conference on Information and knowledge management*. 381–388.

[33] Xinying Song, Jing Liu, Yunbo Cao, Chin-Yew Lin, and Hsiao-Wuen Hon. 2010. Automatic extraction of web data records containing user-generated content. In *Proceedings of the 19th ACM international conference on Information and knowledge management*. 39–48.

[34] Xinying Song, Jing Liu, Yunbo Cao, Chin-Yew Lin, and Hsiao-Wuen Hon. 2010. Automatic extraction of web data records containing user-generated content. In *Proceedings of the 19th ACM international conference on Information and knowledge management*. 39–48.

[35] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. 2016. *Introduction to data mining*. Pearson Education India.

[36] Roberto Panerai Velloso and Carina F Dorneles. 2013. Automatic web page segmentation and noise removal for structured extraction using tag path sequences. *Journal of Information and Data Management* 4, 3 (2013), 173–173.

[37] Roberto Panerai Velloso and Carina F Dorneles. 2017. Extracting records from the web using a signal processing approach. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 197–206.

[38] Jingjing Wang, Haixun Wang, Zhongyuan Wang, and Kenny Q Zhu. 2012. Understanding tables on the web. In *International Conference on Conceptual Modeling*. Springer, 141–155.

[39] Pengfei Wang, Jiafeng Guo, Yanyan Lan, Jun Xu, Shengxian Wan, and Xueqi Cheng. 2015. Learning Hierarchical Representation Model for NextBasket Recommendation. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Santiago, Chile) *(SIGIR '15)*. Association for Computing Machinery, New York, NY, USA, 403–412. https://doi.org/10.1145/2766462.2767694

[40] Xiaoqin Xie, Yixiang Fang, Zhiqiang Zhang, and Li Li. 2012. Extracting data records from web using suffix tree. In *Proceedings of the ACM SIGKDD workshop on mining data semantics*. 1–8.

[41] Yasuhiro Yamada, Nick Craswell, Tetsuya Nakatoh, and Sachio Hirokawa. 2004. Testbed for information extraction from deep web. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*. 346–347.

[42] Mohammed J Zaki and Wagner Meira Jr. 2020. *Data Mining and Machine Learning: Fundamental Concepts and Algorithms*. Cambridge University Press.

[43] Yanhong Zhai and Bing Liu. 2005. Web data extraction based on partial tree alignment. In *Proceedings of the 14th international conference on World Wide Web*. 76–85.

[44] Yanhong Zhai and Bing Liu. 2006. Structured data extraction from the web based on partial tree alignment. *IEEE Transactions on Knowledge and Data Engineering* 18, 12 (2006), 1614–1628.

[45] Zhixian Zhang, Kenny Q Zhu, Haixun Wang, and Hongsong Li. 2013. Automatic extraction of top-k lists from the web. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE, 1057–1068.

[46] Hongkun Zhao, Weiyi Meng, Zonghuan Wu, Vijay Raghavan, and Clement Yu. 2005. Fully automatic wrapper generation for search engines. In *Proceedings of the 14th international conference on World Wide Web*. 66–75.