Web Record Extraction with Invariants

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

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

Eduard Dragut
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 **Miria** (**Mi**ning **r**ecord **invaria**nt), a bottom-up, recursive approach to construct the Web records from the invariants. The proposed approach is both effective and efficient, consistently outperforming the state-of-the-art Web record extraction methods on modern Web pages.

PVLDB Reference Format:
Zhijia Chen, Weiyi Meng, and Eduard Dragut. Web Record Extraction with Invariants. PVLDB, 16(4): 959 - 972, 2022.
doi:10.14778/3574245.3574276

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/ZhijiaCHEN/Web-Record-Extraction-with-Invariants.

1 INTRODUCTION

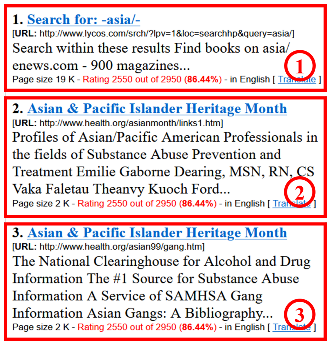
Structured data is an important type of information on the Web. Such data is often in the form of records retrieved from underlying databases and displayed on Web pages according to some templates [13, 28]. We follow past literature and call them *data records* [59]. Mining data records on the Web is useful because it enables one to integrate data from multiple Web sites and provide value-added services, e.g., comparative shopping and meta-querying [14–17, 19]. There is a rich literature on extracting data records from Web pages, dating back to the early days of the Web [21]. With very few exceptions that focus on HTML template extraction [41], the fully automatic approaches reported in the literature make, either explicitly or implicitly, two key assumptions: (1) uniform schema assumption – that records are drawn from a database with a uniform schema and (2) linear structure assumption – 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 path* (the tag sequence from the root of the DOM tree to a node) [22, 40, 56], subtree [1, 48, 59], and sibling nodes [50]. However, similarity-based methods do not work well on records with heterogeneous schemas or nesting structures. With the advent of Web 2.0, the structure variations of Web records have become much more common, making existing record extraction methods less effective on modern Web pages. As we will report in Section 5.3.1, the performance of several existing methods varies between 45% and 94% on modern Web pages.

To ease discussion 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 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.

Consider results produced by search engines. Twenty years ago, results produced by search engines in response to a query were a list of records wrapped in a uniform structure with metadata fields such as page title, 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 provide data fields according to the media type of its content. As an illustration, Figure 1b shows three 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 on the page; the second is for an encyclopedia page which contains some questions and their answers about Einstein; 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 shown in 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 via comments and replies (a reply to a comment becomes the child of that comment) and then borrowed by Web sites to render 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 1c where record 1 and record

Proceedings of the VLDB Endowment, Vol. 16, No. 4 ISSN 2150-8097.
doi:10.14778/3574245.3574276

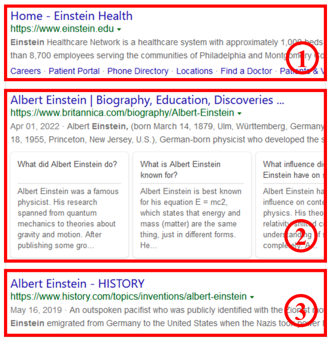


1. Search for: -asia/-
[URL: http://www.lycos.com/srch/?lpv=1&loc=searchhp&query=asia/]
Search within these results Find books on asia/ enews.com - 900 magazines...
Page size 19 K - Rating 2550 out of 2950 (86.44%) - in English [Translate]

2. Asian & Pacific Islander Heritage Month
[URL: http://www.health.org/asianmonth/links1.htm]
Profiles of Asian/Pacific American Professionals in the fields of Substance Abuse Prevention and Treatment Emilie Gaborne Dearing, MSN, RN, CS Vaka Faletau Theanvy Kuoch Ford...
Page size 2 K - Rating 2550 out of 2950 (86.44%) - in English [Translate]

3. Asian & Pacific Islander Heritage Month
[URL: http://www.health.org/asian99/gang.htm]
The National Clearinghouse for Alcohol and Drug Information The #1 Source for Substance Abuse Information A Service of SAMHSA Gang Information Asian Gangs: A Bibliography...
Page size 2 K - Rating 2550 out of 2950 (86.44%) - in English [Translate]

(a) TBDW dataset example [57].



Home - Einstein Health
https://www.einstein.edu
Einstein Healthcare Network is a healthcare system with approximately 1,000 beds than 8,700 employees serving the communities of Philadelphia and Montgomery Co
Careers · Patient Portal · Phone Directory · Locations · Find a Doctor · Patients & V

Albert Einstein | Biography, Education, Discoveries ...
https://www.britannica.com/biography/Albert-Einstein
Apr 01, 2022 · Albert **Einstein**, (born March 14, 1879, Ulm, Württemberg, Germany 18, 1955, Princeton, New Jersey, U.S.), German-born physicist who developed the s

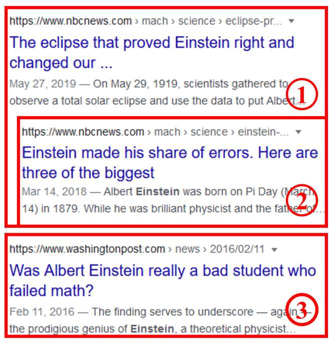
What did Albert Einstein do?
Albert Einstein was a famous physicist. His research spanned from quantum mechanics to theories about gravity and motion. After publishing some gro...

What is Albert Einstein known for?
Albert Einstein is best known for his equation E = mc2, which states that energy and mass (matter) are the same thing, just in different forms. He...

What influence di Einstein have on
Albert Einstein ha influence on conte physics. His theo relativity shifted c understanding of s completely. A...

Albert Einstein - HISTORY
https://www.history.com/topics/inventions/albert-einstein
May 16, 2019 · An outspoken pacifist who was publicly identified with the Zionist mo **Einstein** emigrated from Germany to the United States when the Nazis took power i

(b) Search results from Bing.

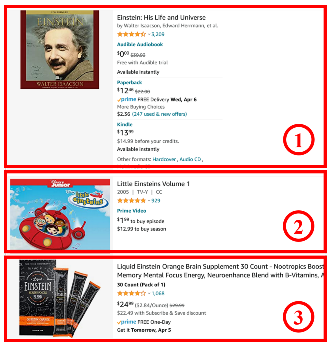


https://www.nbcnews.com › mach › science › eclipse-pr...
The eclipse that proved Einstein right and changed our ...
May 27, 2019 — On May 29, 1919, scientists gathered to observe a total solar eclipse and use the data to put Albert...

https://www.nbcnews.com › mach › science › einstein-...
Einstein made his share of errors. Here are three of the biggest
Mar 14, 2018 — Albert **Einstein** was born on Pi Day (March 14) in 1879. While he was brilliant physicist and the father of...

https://www.washingtonpost.com › news › 2016/02/11
Was Albert Einstein really a bad student who failed math?
Feb 11, 2016 — The finding serves to underscore — again — the prodigious genius of **Einstein**, a theoretical physicist...

(c) Search results from Google.



Einstein: His Life and Universe
by Walter Isaacson, Edward Herrmann, et al.
3,209
Audible Audiobook
$0.00 ~~$39.93~~
Free with Audible trial
Available instantly
Paperback
$12.46 ~~$22.00~~
prime FREE Delivery **Wed, Apr 6**
More Buying Choices
$2.36 (247 used & new offers)
Kindle
$13.99
$14.99 before your credits.
Available instantly
Other formats: Hardcover , Audio CD ,

Little Einsteins Volume 1
2005 | TV-Y | CC
929
Prime Video
$1.99 to buy episode
$12.99 to buy season

Liquid Einstein Orange Brain Supplement 30 Count - Nootropics Boost Memory Mental Focus Energy, Neuroenhance Blend with B-Vitamins, A
30 Count (Pack of 1)
1,068
$24.99 ($2.84/Ounce) ~~$29.99~~
$22.49 with Subscribe & Save discount
prime FREE One-Day
Get it **Tomorrow, Apr 5**

(d) Search results from Amazon.

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

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) [50] assumes that data records on a page must share some attributes, such as the posting date of a comment. 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 [22], 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, this method only considers linear records, and the prefix tree cannot handle nested records. There are also solutions that focus on records with user-generated content [26, 27], such as comments and postings, and thus they can leverage domain-specific features to help locate records and determine boundaries [5, 11, 29]. However, these are not generic record extraction solutions.

Despite the heterogeneity and non-linearity of Web records, Web records on the same result page still exhibit common elements. For example, the three records in Figure 1d all have a rating and a price. Our solution is predicated on the presence of common record elements which we refer to as *record invariants*. We observe two types of invariants, and here we illustrate them with the running example shown in Figure 2. *We use $\mathbb{E}$ to denote the DOM tree of this example for the rest of this paper.* The example contains three records that present the challenges we mentioned previously: Record 1 contains an image, while Record 3 is nested under Record 2. But still, one can easily observe some common structures among the three records: The titles are expressed with subtrees (under nodes $i4$, $i18$, and $i26$) of identical structures. We call the occurrence of such a pattern as **invariant subtree**. Moreover, we observe that the paths from the occurrences of the invariant subtree to the corresponding records' top nodes ($i2$, $i16$, and $i24$) have the same sequence of HTML tags. We call the occurrence of such a path pattern as **invariant path**. The invariants may be part of the data (e.g., a common attribute) or of the rendering HTML template. Our goal is to mine such invariants automatically and use them to reconstruct the records.

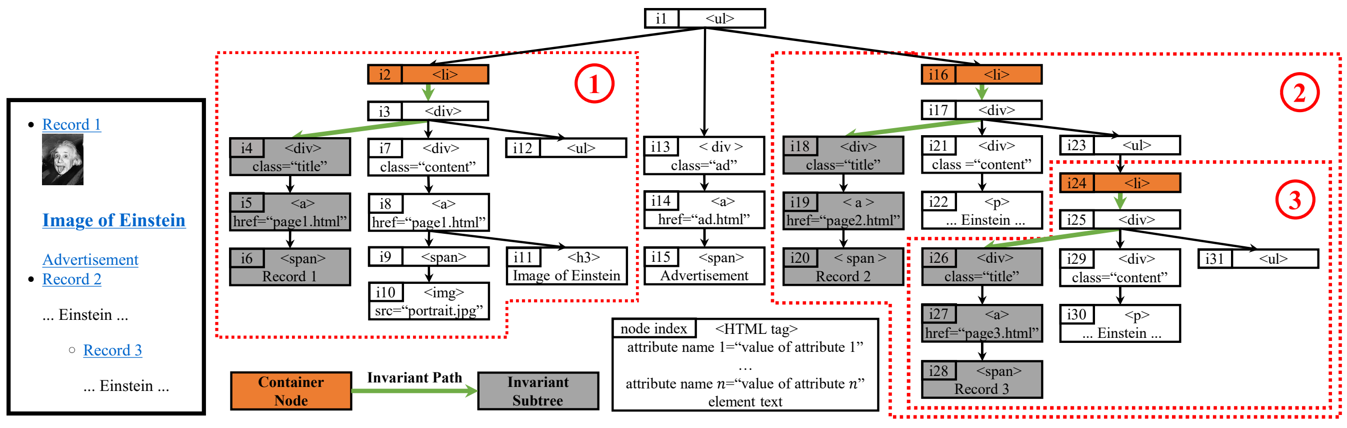
To this end, we propose **Miria** (**Mi**ning **r**ecord **i**nvari**a**nt), a bottom-up algorithm that turns a DOM tree into a sequence to search for invariant subtrees based on frequent sequential patterns and then recovers the Web records by matching invariant paths. Extracting Web records based on sequential pattern matching has been explored in several works [8, 22, 40, 53]. However, they all require Web records to observe the uniform schema and linear structure assumptions. More specifically, most of these methods encode a node using its tag path [22, 40, 53], which hampers their ability to detect patterns among nested Web records because their tag paths can be very different. In contrast, we propose to encode a node using the *subtree structure under a node*, eliminating the variation introduced by nested structures (the same subtree structure will have the same code value). Furthermore, to recover a record from an occurrence of a pattern, existing methods involve some similarity measurements on the encoding sequence [8, 22], which is sensitive to structure variations. In contrast, Miria tries to align Web records by their invariant subtrees and invariant paths, avoiding enforcing record-level similarity comparison. The proposed algorithm achieves accurate and consistent performance across multiple datasets, both old and new, empirically proving that it can cope with diverse types of Web records on the modern Web.

We make the following contributions in this work:

- We show that the assumptions followed by traditional Web record extraction tools are outdated on the modern Web and propose new ones that are more reflective of today's Web.
- We define Web record invariants and propose novel algorithms using methods inspired by signal processing literature to mine such invariants from Web pages.
- We propose a bottom-up approach for mining Web data records. To the best of 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 existing approaches for Web record extraction can be classified by the level of automation: manual approaches, semi-automatic approaches, and automatic approaches.



(a) Rendered page. **(b) The DOM tree of this example, denoted as $\mathbb{E}$ throughout this paper.**

Figure 2: A running example of three Web records with heterogeneous contents and nested structures. Record boundaries are marked with red dotted lines.

Manual approaches aim at writing specialized tools to extract Web records from a limited number of Web pages [18, 33]. They require manual inspection of Web pages and their source code. Then one writes a specialized program in the form of overfitted rules (e.g., XPath 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.

Semi-automatic approaches usually involve supervised or semi-supervised learning as they require human-labeled examples to learn extraction rules and synthesize wrappers for annotated websites [7, 34, 44]. These approaches are generally centered around the topic of wrapper induction, where the major challenges include the efficient generation of wrappers [32, 62], the robustness against structural variation of website templates [42, 47], and the maintenance of the wrappers when websites change templates [35, 43].

The automatic approaches focus on detecting frequent patterns based on the fact that Web records from the same logical table share very similar structure patterns both visually (as rendered on the target Web page) and in their underlying HTML source codes. Automatic solutions target two extraction scenarios in general: extract multiple list-like records from a single Web page and extract one record from a detailed page with access to multiple pages of the same template [6, 41]. This paper is focused on the former, and we present a number of relevant works here.

There are HTML tags that are designed to format lists in Web pages, such as `<ul>` and `<ol>`. Thus, some works are dedicated to extracting records using these tags [3, 10, 20, 45, 54, 61], but they do not generalize well across the Web. Most approaches apply various pattern mining algorithms to locate record regions within a page by analyzing the DOM trees, such as repeating tags [2, 21], repeating tag paths [8, 22, 24, 30, 40, 52], and repeating subtrees [36, 59]. 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 determining the record boundaries. Existing 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 [52], sibling alignment [50], or tree alignment [37, 59, 60]. The similarity may also be performed on a sequential data structure, where the DOM tree is transformed into a sequence/string [8, 22, 40, 53]. Reducing tree data structures to sequence data structures introduces the opportunity of applying sequential pattern mining techniques to the Web record extraction problem, and our method also follows this approach.

Apart from the DOM tree structure, some works leverage the visual layout in the rendered Web page [4, 23, 49]. There are also hybrid methods that combine the two [38, 63]. 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 show that these assumptions no longer hold for modern Web pages. Finally, we revise these 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.

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. We examine these assumptions below and explain with supporting examples why it is violated on the modern Web.

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.

The "continuous" condition in Assumption 1 was almost universally satisfied in Web 1.0, where the ad industry had yet to understand the full potential of digital advertisement [39]. The common violation of the "continuous" assumption is the (strategic) insertion of ads between 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 [55]. Thus, most Web sites today contain 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.

Informally, the "almost" condition in Assumption 2 is a consequence of the similarity condition in Assumption 1. If data records follow the same schema, 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 – 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. For instance, the records of Einstein Health and Britannica returned by Bing in response to "Einstein" (Figure 1b) have different sets of attributes: the record of Einstein Health contains the main navigation links of the website, such as "Career" and "Patient Portal", whereas the record of Britannica contains several questions and answers about Einstein. This is because Bing is able to distinguish between the records' industry types 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 specific semantics. For example, in social media, a reply to a post becomes the child of that post, and search engines like Google use it to combine records on a topic retrieved from the same Web site, as shown in Figure 1c 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 [25]. For example, Google's Big Table [9] and Amazon's DynamoDB [12] are not relational and can better accommodate records with diverse schemas.

index	tag path	tag path code
i1	ul	1
i2	ul/li	2
i3	ul/li/div	3
…	…	…
i16	ul/li	2
…	…	…
i31	ul/li/ul/li/div/ul	20

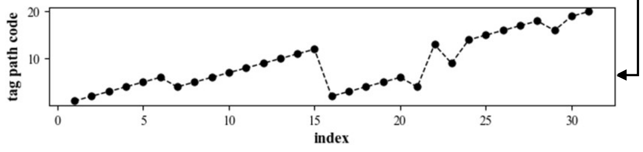


Figure 3: Tag path code sequence of $\mathbb{E}$ (Figure 2b).

3.2 A Picture is Worth a Thousand Words

The previous section provided a qualitative analysis of the differences between Web 1.0 and Web 2.0 records. In this section, we give a quantitative argument of the changes triggered by the violations of those assumptions.

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 draw inspiration from visual data mining with signal processing techniques [31]. Intuitively, suppose we can map Web records to cycles in a signal. If these assumptions are well satisfied, then 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 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 the nodes of $\mathbb{E}$. 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 [22, 40, 53].

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 [57], which was collected in 2003 and widely used as a benchmark dataset for Web record extraction (see more details in Section 5.1). We randomly choose 1 page from three Web sites (1, 25, and 50) that are at the beginning, the middle, and the end of the dataset. As representatives of Web 2.0 records, we collected three datasets of Web records in 2021 from Google, Amazon, and news comments, respectively (see Section 5.1 for details). We take a random page from each of them. Note that we do not plot the sequential signal of the entire Web pages but only that of the main data region.

Figure 4 shows 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 the TBDW dataset have neat repeating patterns, indicating that there is little structure variation among records. Specifically, the

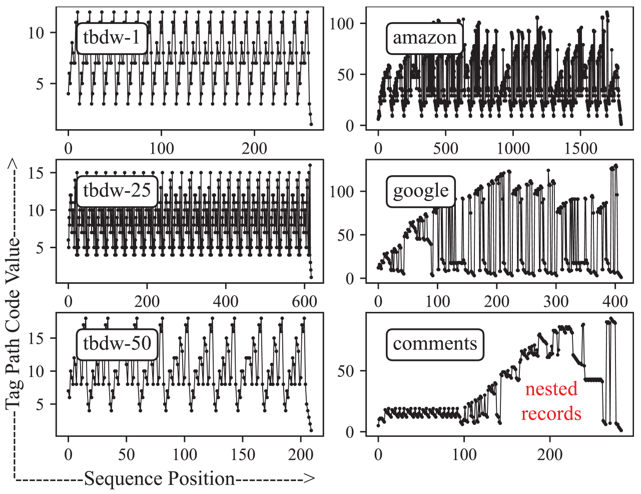


Figure 4: Signals of tag path codes of Web 1.0 (left column) and Web 2.0 records (right column).

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 modern Web pages. Consider the examples from Google and comments which contain an arch in their sequences. This indicates the presence of nested records: Web records under the same parent node will share some common tag path codes, but for those nested Web records, their tag path codes are very different from the non-nested ones. Such behavior is not encountered in the samples from the TBDW dataset.

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 catch up with the complex HTML structures on the modern Web. For that, we need to distinguish between a *data record* and a *Web record*. A *Web record* has two parts: the data part, which includes the HTML tags that directly enclose the *data record* retrieved from a database, and the *non-data* part, which includes the tags for interaction purposes (e.g., Add to Cart button) and the tags for formatting purpose. For example, the data part of the second record in Figure 1d includes the tags that contain the cover image, "Little Einsteins Volume 1", "2005 | TV-Y | CC", "929", "$1.99", and "$12.99"; the non-data part includes the tags that form the product ratings (which will expand into the distribution of ratings when the mouse pointer is hovering on them) and all the non-displayable tags that help format 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 some common subtree structures. If one is able to identify such subtrees, then we contend that one is able to reconstruct the Web records and extract the data with high accuracy. We call such a subtree an *invariant subtree.*

Definition 3.1 (Invariant Subtree)**.** An invariant subtree is a subtree structure that presents and serves the same purposes in every Web record from the same logical table.

Invariant subtrees 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. A Web record template generally represents the same attribute using the same tag sequence.

The second level is the non-data part, which usually follows a template that is not sensitive to the type of data record. For instance, records in 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 1c, all the page titles are formatted with a `<h3>` tag along with an `<a>` tag. The invariant subtrees from the non-data part help us recover Web records even if their data records are from 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 more (non-trivial) invariant subtrees.

Recall from the previous section that the presence of nested records in data regions is the main offender of Assumption 3. The reason is that nested records in a data region can form a tree of an arbitrary depth 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 as 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, and we refer to the root node of a Web record as its *container node.*

Definition 3.2 (Container Node)**.** The container node of a Web record is the highest node in a DOM tree such that all the data fields of this record are descendants of the node, and no data fields of other records are among its descendants unless the record is nested.

For example, nodes $i2$, $i16$, and $i24$ of $\mathbb{E}$ are the container nodes of the three records. Note that node $i16$ is the container node of Record 2, and it only contains the data fields of Record 2 and Record 3, but the latter is nested under the former. Using Web record container nodes as reference points, we can define *invariant path.*

Definition 3.3 (Invariant Path)**.** For Web records in the same data region, there exists one invariant subtree such that the paths from the record container nodes to the occurrences of the invariant subtree have the same sequence of tags for all records. We call the occurrence of such a repeating path an invariant path.

For example, the three occurrences of the invariant subtree in $\mathbb{E}$ have the same tag path to their container nodes, i.e., `<div>`, `<li>`. Note that even though Record 3 is nested under Record 2, its path from its container node to its invariant subtree is the same as that of Record 2. There may be more than one invariant subtree present among a set of Web records, and we only require one of them to have an invariant path. We replace Assumption 3 with:

Assumption 5. Web records in a data region of a Web page contain one or more invariant paths.

Finally, to cope with the "continuous" requirement in Assumption 1 in the presence of breaks, we assume that a data region is a collection of Web records sharing invariant subtrees and invariant paths. This allows us to omit ad sections or other types of sections inserted between the data region's records. 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. This does not include wrapper induction approaches which can be both top-down and bottom-up, and more recent ones are hybrid [47], combining the benefits of the two approaches.

3.4 Notations

Before we dive into the details of our solution, we introduce a few notations that help the presentation. 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.

4 APPROACH

With Assumptions 4 and 5, we present Miria, a bottom-up Web record extraction algorithm that detects records by first finding invariant subtrees and then matching invariant paths to the corresponding record container nodes.

We provide an overview of Miria 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 (Section 4.1).
(2) Encode each node into an integer with an appropriate scheme (Section 4.2).
(3) Search invariant subtrees by mining frequent patterns on the code sequence (Section 4.3).
(4) Align Web records vertically by matching the invariant paths from the invariant subtrees to record container nodes (Section 4.4).
(5) If there are gaps among sibling Web records, align them horizontally by matching their preceding and following subtrees (Section 4.5).

4.1 Flatten The Tree

We start by converting 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.

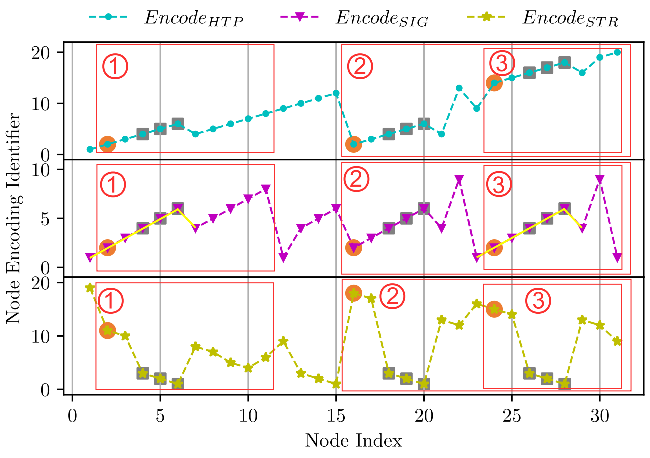


Figure 5: The NES of $\mathbb{E}$ (Figure 2b), encoded by $Encode_{HTP}$, $Encode_{SIG}$, and $Encode_{STR}$, respectively. The red rectangles show the boundary of the records. The record container nodes are marked by orange circles, and the nodes of invariant subtrees are marked by gray squares. The bright yellow line in the middle figure highlights a pattern that spans over the record boundary.

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 (NES))**.** The NES of a tree D is a sequence of codes obtained by a node encoding function. We assign a positive 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 NES Construction

Our definition of NES is a generalization of the **Web Page Sequence (WPS)** [22] 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 $\mathbb{E}_{i6}$ is:

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

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, we hope the three occurrences of the invariant subtree of $\mathbb{E}$ have the same code, but that cannot be satisfied if we encode a node using its tag path (see Figure 3). This situation motivates the problem: how to encode a node so that all occurrences of an invariant subtree 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 `<div>` and `<span>`, 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. 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 $\mathbb{E}_{i3}$ and $\mathbb{E}_{i4}$ are:

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

We use attribute names instead of attribute values because we want similar nodes to have the same encoding, and often attribute values are unique to a node, such as `id` and `href`. For example, if we use attribute values, then $Encode_{SIG}(\mathbb{E}_{i5})$, $Encode_{SIG}(\mathbb{E}_{i19})$, and $Encode_{SIG}(\mathbb{E}_{i27})$ would not have the same encoding value because their `href` values are different. Apart from encoding a node by its signature, we also propose encoding a node by its structure. In fact, when we discuss similar structures of records, we implicitly assume 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 recursively as:

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

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 and text length.

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 $\mathbb{E}$ 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, where the record container nodes are marked by orange circles, and the nodes of invariant subtrees are marked by gray squares. 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 three records completely or partially matched. However, the three occurrences of the invariant subtree ($\mathbb{E}(i4)$, $\mathbb{E}(i18)$, and $\mathbb{E}(i26)$), the content snippet field of Record 1 ($\mathbb{E}(i7)$), and the advertisement ($\mathbb{E}(i13)$), all share the same code sequence: $\langle 4, 5, 6 \rangle$. 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 $\langle 3, 2, 1 \rangle$ as the most frequent pattern, and it only matches with the invariant subtree in each record ($\mathbb{E}(i4)$, $\mathbb{E}(i18)$, and $\mathbb{E}(i26)$) and the advertisement ($\mathbb{E}(i13)$). The advertisement will be discarded later when matching the invariant path.

It is obvious that if the target Web records share at least one invariant subtree, 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 main reason 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). Given a DOM tree D, a node encoding function f is a constant node encoding function if $Encode_f(D)$ is determined only by nodes in D.

Lemma 1. If Web records share at least one invariant subtree, 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 (Equation 5) or the corresponding subtree size (Equation 6). Moreover, the chance of layout equivalence is smaller than depth equivalence; thus, structure encoding is expected to perform better in pattern precision.

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 construction 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)
    sig ← node signature of node
    if sig ∉ sigIDMap then
        sigIDMap(sig) ← |sigIDMap|
    sigId ← sigIdMap(sig)
    struc ← empty array
    struc.append(sigId)
    foreach child ∈ children of node do
        struc.append(structure encoding ID of child)
    if struc ∉ strucIdMap then
        strucIdMap(struc) ← |strucIdMap|
    strucId ← strucIdMap(struc)
    NES.append(strucID)
```

4.3 Frequent Pattern Detection

4.3.1 Pattern Mining. With Assumption 4 and Lemma 1, we have a systematic way to locate Web records by detecting frequent substrings (i.e., continuous subsequences) in an NES constructed by a constant encoding function. The frequent substring mining problem is a classic 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. 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 [58]. For illustration, we built a suffix tree for the $Encode_{SIG}$ NES of $\mathbb{E}$ (the second sub-figure 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 edge is labeled by a substring. Concatenating the edge 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 four substrings starting with $\langle 1 \rangle$ and three substrings starting with $\langle 2, 3, 4, 5, 6, 4 \rangle$. 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 the edges on its path to the root is greater than L_{th} and 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: $\langle 1, 2, 3, 4, 5, 6, 4 \rangle$, $\langle 2, 3, 4, 5, 6, 4 \rangle$, $\langle 2, 3, 4, 5, 6, 4, 9, 1 \rangle$, $\langle 3, 4, 5, 6, 4 \rangle$, $\langle 3, 4, 5, 6, 4, 9, 1 \rangle$.

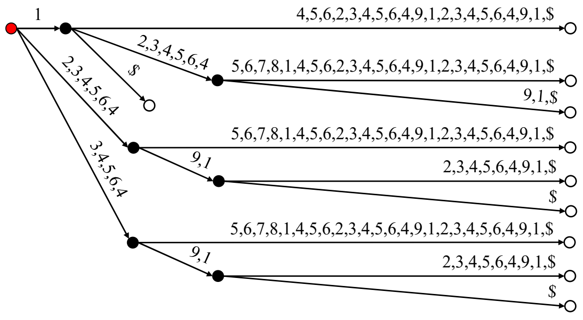


Figure 6: The first three branches of the suffix tree constructed from the $Encode_{SIG}$ NES of Figure 5

One notices that there may be redundant patterns because some patterns are substrings 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.

Take the above patterns as an example, $< 1, 2, 3, 4, 5, 6, 4 >$ (with two supports) is a super-pattern of $< 2, 3, 4, 5, 6, 4 >$ (with three supports), and both are closed patterns because they do not have any super-pattern with equal support. The pattern $< 3, 4, 5, 6, 4 >$ (with three supports), however, is not a closed pattern because it has a super-pattern $< 2, 3, 4, 5, 6, 4 >$ that has equal supports of 3.

A closed pattern is a minimal representation of its super-patterns without losing their support information [51]. 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: $\langle 1, 2, 3, 4, 5, 6, 4 \rangle$, $\langle 2, 3, 4, 5, 6, 4 \rangle$, $\langle 2, 3, 4, 5, 6, 4, 9, 1 \rangle$.

4.3.2 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 across the boundary of records**, which makes it difficult, if not impossible, to decide the correct record container node in our following step. For example, the closed pattern $\langle 1, 2, 3, 4, 5, 6, 4 \rangle$ has two occurrences in the $Encode_{SIG}$ NES of Figure 5 (highlighted by yellow lines), starting at indexes 1 and 23, respectively, and they span over the boundaries of the corresponding records (marked in red rectangles). 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 a 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 2.

Lemma 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 $p' \leftarrow \emptyset$;
2 for $i$:=1 to length($p$) step 1 do
3     for $j$:=length($p$) to $i$ step -1 do
4         $O \leftarrow$ the occurrences of $p[i:j]$;
5         if $|\{LCA(oi), oi \in O\}| = |O|$ then
6             if $length(p[i:j]) > length(p')$ then
7                 $p' \leftarrow p[i:j]$;
8             break;
9 return $p'$;
```

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  $C \leftarrow A$;
2  while $|C| > 0$ do
3      $P \leftarrow \emptyset$;
4      foreach $c \in C$ do
5          $p \leftarrow$ the parent node of $c$;
6          if $\forall y \in C, y \subset p$ then
7              return C;
8          add $p$ to $P$;
9      $G \leftarrow P$ divided into groups such that the nodes in the
       same group have the same signature;
10     $C \leftarrow$ the largest group of $G$;
11 return $C$;
```

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

2) r_{oi} and r_{oj} are two different Web 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 $LCA(oi) \neq LCA(oj)$. □

Lemma 2 gives the sufficient but not the necessary condition for patterns that span multiple records. In practice, we found that the condition is 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 satisfies our condition. After pattern reduction, we apply the pattern length threshold one more time to evict trivial patterns produced by the reduction process.

Applying Algorithm 2, the above three closed patterns are reduced to $\langle 2, 3, 4, 5, 6, 4 \rangle$, $\langle 2, 3, 4, 5, 6, 4, 9, 1 \rangle$. 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 $\langle 2, 3, 4, 5, 6, 4 \rangle$ is the final frequent pattern for this example.

4.4 Vertical Alignment

Our last step is to find record container nodes using the pattern candidates. We use a pattern's occurrences as anchors to align Web records by matching potential invariant paths. 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 $Encode_{STR}(\mathbb{E})$ in Figure 5 as an example (yellow dashed line with star markers), the pattern $\langle 3, 2, 1 \rangle$ has 4 occurrences starting at indexes 4, 13, 18, and 26, respectively. By mapping the indexes back to $\mathbb{E}$ in Figure 2b, we see that the corresponding anchor trees are $\mathbb{E}(i4)$, $\mathbb{E}(i13)$, $\mathbb{E}(i18)$, and $\mathbb{E}(i26)$. An anchor tree is either derived from an invariant subtree of a Web record or some background noise that happens to have the same structure as an invariant subtree, such as $\mathbb{E}(i13)$ in this case.

By Assumption 5, we can find the record container nodes by matching the ancestors of anchor trees. 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). Note that a group of anchor trees is not necessarily formed by an invariant subtree, and an invariant path may not exist for the group. Thus when we group them by their 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).

For example, if we use the frequent pattern $\langle 3, 2, 1 \rangle$ from the NES of $Encode_{STR}(\mathbb{E})$, then the record container nodes will be initialized by $\langle \mathbb{E}_{i4}, \mathbb{E}_{i13}, \mathbb{E}_{i18}, \mathbb{E}_{i26} \rangle$. In the first iteration, the container nodes are updated to $\langle \mathbb{E}_{i3}, \mathbb{E}_{i1}, \mathbb{E}_{i17}, \mathbb{E}_{i25} \rangle$. They will be split into two groups: $\langle \mathbb{E}_{i3}, \mathbb{E}_{i17}, \mathbb{E}_{i25} \rangle$ and $\langle \mathbb{E}_{i1} \rangle$ because $\mathbb{E}_{i1}$ has a different node signature from the others, and we select the larger group. In the next iteration, the group is updated to $\langle \mathbb{E}_{i3}, \mathbb{E}_{i17}, \mathbb{E}_{i25} \rangle$. In the last iteration, the parent node of $\mathbb{E}_{i2}$, i.e., $\mathbb{E}_{i1}$, is the lowest common ancestor of all the container nodes, and thus we exit the loop. We return the final record container nodes $\langle \mathbb{E}_{i2}, \mathbb{E}_{i16}, \mathbb{E}_{i24} \rangle$.

4.5 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. For example, a record from a search engine may contain one subtree for the page title, followed by another subtree displaying the page content snippet. To handle this case, we add an optional step of horizontal alignment to include the preceding and following subtrees for the Web 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

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

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.

4.6 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 occurrences per pattern, 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 alignment 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 $H \cdot N_p \cdot N_o$, which is bounded by $O(HN)$ because $N_p \cdot N_o < N$. Overall, the running time of Miria has a loose upper-bound of $O(HN)$. In practice, DOM trees tend to grow horizontally rather than vertically for data regions. For example, the DOM trees of the result Web pages of a search engine have very similar tree heights irrespective of the number of records in each Web page. In the five datasets we used in this paper (Section 5.1), their average tree heights are similar (ranging from 10 to 20) while their average tree sizes are quite different (ranging from 400 to 5000). Hence, if we regard H to be a constant or a small integer, the solution has an almost linear running time of $O(N)$ in practice. This is confirmed by the efficiency experiments in Section 5.4.

5 EXPERIMENTS

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

We implement the method in Python and 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 Miria 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 [57]. It includes 114,540 results pages from 51 deep Web sites. It was created in 2003 and still serves as a benchmark. For Web 2.0 records, we use a more recent dataset **EX**, which combines the EX1 and EX2 datasets from [47]. The two datasets were annotated for record attribute extraction (e.g., title, price). We annotated the corresponding record container nodes of the target attributes and filtered out duplicate pages. EX contains 82 pages from 72 websites across various domains, but it does not contain nested Web records, and its records do not present significant structure variation. Thus for this study, we created more challenging datasets that represent i) the latest Web programming paradigms and ii) Web pages with nested regions. We constructed three 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 regular 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 ten comments, uniformly distributed over 100 Web sites.

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

5.2 Baselines

We implement Miria with each of the three node encoding schemes introduced in Section 4.2, and we compare it with four baselines: **DEPTA** [59] is a widely referenced baseline in the literature. **MiBAT** [50] 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 and EX since they are collected from multiple domains, and we are not able to define an invariant pattern that works for all of them. The methods proposed in **Velloso et al.** [53] and **PROSE** [46] are chosen as representatives of the state-of-the-art solutions. **PROSE** [46] is a program synthesis API from Microsoft. The API (www.microsoft.com/en-us/research/group/prose/) allows

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)
EX	72	82	4814	1.14 (0.39)	58.71 (188.09)	3429.95 (5665.83)	12.42 (4.87)
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)

Table 2: Precision (P), recall (R), and F1 for the Web record extraction task.

method		TBDW			EX			AMAZON			GOOGLE			COMMENT		
		R	P	F1	R	P	F1	R	P	F1	R	P	F1	R	P	F1
Miria	$Encode_{SIG}$	0.92	0.87	0.89	**0.97**	0.82	0.87	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	0.95	0.88	0.91	**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	0.92	**0.93**	0.95	0.95	0.95	0.93	0.93	**0.93**	0.96	0.95	**0.95**
DEPTA [59]		0.89	0.99	0.94	0.79	0.95	0.86	0.61	**0.98**	0.75	0.30	0.91	0.45	0.42	0.94	0.58
Velloso et al. [53]		0.94	0.92	0.93	0.78	**0.97**	0.86	0.80	0.94	0.87	0.41	0.90	0.56	0.47	0.90	0.61
MiBAT [50]		n/a	n/a	n/a	n/a	n/a	n/a	0.95	0.93	0.94	0.49	**1.00**	0.66	0.66	0.99	0.79
PROSE [46]		**0.99**	**1.00**	**0.99**	0.89	0.93	0.91	0.88	**0.98**	0.93	0.94	0.91	0.92	0.77	0.88	0.82

users to synthesize the Web table extraction program automatically [46] without any example or semi-automatically with a few examples [47]. If no example is provided, the API will try to infer every potential table from the target page, which is similar to Miria. We compare Miria with this use case.

5.3 Accuracy Analysis

5.3.1 Record Accuracy. In this experiment, we measure the overall accuracy of record extraction. Because a method may detect multiple groups of records in one page, we select the group that has the most overlap with the annotated records. An output record is considered correct if its displayable texts are completely matched with a ground truth record.

Table 2 shows the average record recall, precision, and F1 score. On the Web 1.0 dataset TBDW, the PROSE method has a nearly perfect performance of 0.99 F1 score, followed by Miria with the $Encode_{STR}$ encoding scheme and DETPA that both achieve an F1 score of 0.94. When it comes to Web 2.0 datasets, Miria achieves the best F1 score, with $Encode_{STR}$ winning on the EX, GOOGLE, and COMMENT datasets, and $Encode_{HTP}$ winning on the AMAZON dataset. We draw attention to the $Encode_{HTP}$'s recall: it is near perfect on the EX and AMAZON datasets but relatively poor on the GOOGLE and COMMENT datasets. This is because $Encode_{HTP}$ cannot handle nested records, and there is no nested records in the EX and AMAZON datasets. In contrast, the GOOGLE and the COMMENT datasets contain 4.62% and 40.49% of nested records, respectively. The performance of Miria remains steady across all the datasets, but the baselines generally suffer significant recall losses on the Web 2.0 records. MiBAT achieves a recall of 0.95 on the AMAZON dataset, which can be explained by the non-nested structure of this dataset. It achieves perfect precision on GOOGLE because the invariant pattern we chose appears in all Web records. PROSE also has an outstanding recall on GOOGLE (0.94). Noticeably, the method has a similar performance on the Web 2.0 datasets as Miria. The tiebreaker is the COMMENT dataset, where Miria outperforms PROSE (.95 versus .82 F1-scores). This shows that PROSE does not cope well with 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

5.3.2 Anchor Accuracy. 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 targeted Web record while avoiding producing invalid anchor trees that may introduce false positives. Here we distinguish between the true and false anchor trees: an anchor tree is a true anchor tree if it belongs to a target Web record. Thus, given a group of records R, let A be a set of anchor trees derived from an invariant subtree of R, we define the anchor recall and anchor precision of A by:

$$\text{anchor recall} = \frac{\text{number of true anchor trees in } A}{|R|} \tag{7}$$

$$\text{anchor precision} = \frac{\text{number of true anchor trees in } A}{|A|} \tag{8}$$

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 largest support. $Encode_{STR}$ has the best precision while $Encode_{SIG}$ has the worst, which can be explained by the node encoding collision probabilities given in Equations 4, 5, and 6. As for the recall, both $Encode_{SIG}$ and $Encode_{STR}$ achieve a near-perfect recall 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.

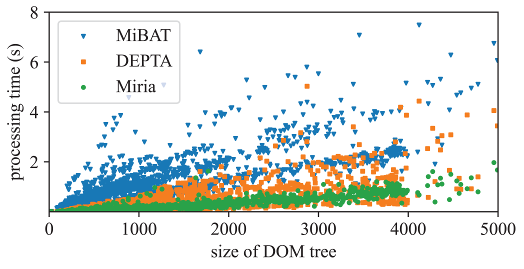


Figure 7: Distribution of processing time vs DOM tree size.

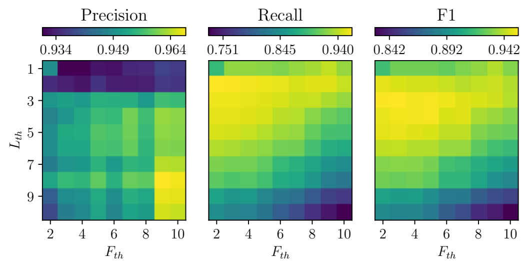


Figure 8: Frequency and pattern length thresholds analysis.

5.4 Efficiency Analysis

We compare the efficiency of Miria (with $Encode_{STR}$) against MiBAT and DEPTA. For fairness, all three methods are implemented in Python. We test the methods on COMMENT and show their distributions of the processing time against the size of the DOM tree in Figure 7. Overall, Miria has the best running time, and it shows a sub-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 attributes with string pattern matching.

5.5 Sensitivity Analysis

We perform the sensitivity analysis of Miria with $Encode_{STR}$ to study how the pattern length and frequency thresholds 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 8. We observe that precision has a strong positive correlation with L_{th} and F_{th} while 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 precision is much less sensitive to the two parameters as the score ranges between 0.930 and 0.968, whereas recall ranges between 0.727 and 0.963. Overall, the best F1 score is 0.954 with L_{th} and F_{th} both equal to 3.

5.6 Case Study

Figure 9 shows 3 synthetic examples that represent the cases where Miria achieves perfect accuracy (a), has false positive (b) and false negative (c), respectively. We use a small *filled* shape to represent a node, and nodes of the same filled shape (despite different colors) have the same node signature. A big *empty* shape represents a

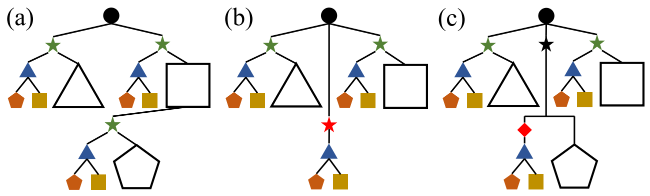


Figure 9: Examples for Miria corresponding to accurate (★ a), false positive (★ b) and false negative (★ c) extraction of record container nodes.

subtree inside a *true* Web record, and different empty shapes represent different subtree structures. The subtrees formed by ⟨▲, ⬟, ■⟩ are the anchor trees detected from frequent patterns, and the star shapes (i.e., ★, ★, ★) are their corresponding container nodes.

Example (a) contains 3 Web records with nested structure. Miria successfully detects all true container nodes (★) while the baselines tend to be confused by nested structures. Example (b) contains 2 Web records with a noise structure (such as a record divider) in the middle, which happens to share a common subtree structure and path with the true Web records. Without any rule to validate a Web record (which is generally domain-specific), Miria would output a false positive (★). Example (c) contains 3 Web records where the anchor tree of the middle one has an extra ◆ on its path to the container node (★), causing Miria to discard the record in the vertical alignment process. Note that this case violates our assumptions about the existence of an invariant path. It may happen when a different style is applied to a Web record, such as an out-of-stock item on a shopping page.

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 into 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 a renewed focus on designing novel Web record extraction methods.

ACKNOWLEDGMENTS

This work was supported in part by U.S. National Science Foundation 1546480, 1546441, and 1838145 grants, and a gift from NVIDIA Corporation.

REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] Valerio Cetorelli, Paolo Atzeni, Valter Crescenzi, and Franco Milicchio. 2021. The smallest extraction problem. *Proceedings of the VLDB Endowment* 14, 11 (2021), 2445–2458.

[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] Yongquan Dong, Eduard C Dragut, and Weiyi Meng. 2018. Normalization of duplicate records from multiple sources. *IEEE Transactions on Knowledge and Data Engineering* 31, 4 (2018), 769–782.

[14] Eduard Dragut, Brian P Beirne, Ali Neyestani, Badr Atassi, Clement Yu, Bhaskar DasGupta, and Meng Weiyi. 2013. YumiInt—A deep Web integration system for local search engines for Geo-referenced objects. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE, 1352–1355.

[15] Eduard Dragut, Wensheng Wu, Prasad Sistla, Clement Yu, and Weiyi Meng. 2006. Merging source query interfaces onweb databases. In *22nd International Conference on Data Engineering (ICDE'06)*. IEEE, 46–46.

[16] Eduard Dragut, Clement Yu, and Weiyi Meng. 2006. Meaningful labeling of integrated query interfaces. (2006).

[17] Eduard C Dragut, Bhaskar Dasgupta, Brian P Beirne, Ali Neyestani, Badr Atassi, Clement Yu, and Weiyi Meng. 2014. Merging query results from local search engines for georeferenced objects. *ACM Transactions on the Web (TWEB)* 8, 4 (2014), 1–29.

[18] Eduard C Dragut, Thomas Kabisch, Clement Yu, and Ulf Leser. 2009. A hierarchical approach to model web query interfaces for web source integration. *Proceedings of the VLDB Endowment* 2, 1 (2009), 325–336.

[19] Eduard C Dragut, Weiyi Meng, and Clement T Yu. 2012. Deep web query interface understanding and integration. *Synthesis Lectures on Data Management* 7, 1 (2012), 1–168.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] Lihong He, Chao Han, Arjun Mukherjee, Zoran Obradovic, and Eduard Dragut. 2020. On the dynamics of user engagement in news comment media. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 10, 1 (2020), e1342.

[27] Lihong He, Chen Shen, Arjun Mukherjee, Slobodan Vucetic, and Eduard Dragut. 2021. Cannot predict comment volume of a news article before (a few) users read it. In *Proceedings of the International AAAI Conference on Web and Social Media*, Vol. 15. 173–184.

[28] Thomas Kabisch, Eduard C Dragut, Clement Yu, and Ulf Leser. 2010. Deep web integration with visqi. *Proceedings of the VLDB Endowment* 3, 1-2 (2010), 1613–1616.

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

[30] 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.

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

[32] Nicholas Kushmerick. 1997. *Wrapper induction for information extraction*. University of Washington.

[33] Alberto HF Laender, Berthier A Ribeiro-Neto, Altigran S Da Silva, and Juliana S Teixeira. 2002. A brief survey of web data extraction tools. *ACM Sigmod Record* 31, 2 (2002), 84–93.

[34] Vu Le and Sumit Gulwani. 2014. Flashextract: A framework for data extraction by examples. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 542–553.

[35] Kristina Lerman, Steven N Minton, and Craig A Knoblock. 2003. Wrapper maintenance: A machine learning approach. *Journal of Artificial Intelligence Research* 18 (2003), 149–181.

[36] 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.

[37] 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.

[38] 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.

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

[40] 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.

[41] 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.

[42] Adi Omari, Sharon Shoham, and Eran Yahav. 2017. Synthesis of forgiving data extractors. In *Proceedings of the tenth ACM international conference on web search and data mining*. 385–394.

[43] Stefano Ortona, Giorgio Orsi, Marcello Buoncristiano, and Tim Furche. 2015. Wadar: Joint wrapper and data repair. *Proceedings of the VLDB Endowment* 8, 12 (2015), 1996–1999.

[44] Oleksandr Polozov and Sumit Gulwani. 2015. Flashmeta: A framework for inductive program synthesis. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*. 107–126.

[45] 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.

[46] Mohammad Raza and Sumit Gulwani. 2017. Automated data extraction using predictive program synthesis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 31.

[47] Mohammad Raza and Sumit Gulwani. 2020. Web data extraction using hybrid program synthesis: A combination of top-down and bottom-up inference. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1967–1978.

[48] 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.

[49] 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.

[50] 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.

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

[52] 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.

[53] 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.

[54] 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.

[55] 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

[56] 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.

[57] 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.

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

[59] 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.

[60] 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.

[61] 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.

[62] 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.

[63] Hongkun Zhao, Weiyi Meng, and Clement Yu. 2006. Automatic extraction of dynamic record sections from search engine result pages. In *Proceedings of the 32nd international conference on Very large data bases*. 989–1000.