Scoring-based DOM Content Selection with Discrete Periodicity Analysis

Thomas Osterland[1,2] and Thomas Rose[1,2]
[1]*Fraunhofer FIT, Schloss Birlinghoven 1, Sankt Augustin, Germany*
[2]*RWTH Aachen University, Ahornstr. 55, Aachen, Germany*

Keywords: Web Scraping, Periodicity Analysis, Content Extraction.

Abstract: The comprehensive analysis of large data volumes forms the shape of the future. It enables decision-making based on empiric evidence instead of expert experience and its utilization for the training of machine learning models enables new use cases in image recognition, speech analysis or regression and classification. One problem with data is, that it is often not readily available in aggregated form. Instead, it is necessary to search the web for information and elaborately mine websites for specific data. This is known as web scraping. In this paper we present an interactive, scoring based approach for the scraping of specific information from websites. We propose a scoring function, that enables the adaption of threshold values to select specific sets of data. We combine the scoring of paths in a web pages DOM with periodicity analysis to enable the selection of complex patterns in structured data. This allows non-expert users to train content selection models and to label classification data for supervised learning.

1 INTRODUCTION

The substantial advances in data analytics, machine learning and stochastic modeling are only a few examples, that demonstrate the importance of data for use cases and applications of the future. The world wide web makes data provision efficient and easily available almost everywhere and also enables the collection of new data on a large scale.

Often data is not readily available in a format, that is easily accessible by computer systems. Web pages are often optimized to improve the visual appeal to humans. The document object model (DOM) structures the content of websites. Thereby, a substantial part of the DOM is not concerned with the direct presentation of information, but with the layout and look to present the information pleasantly, i.e., to improve the presentation such that a human reader can quickly and efficiently identify and extract the most important parts.

Although, this approach is beneficial for human readers, it obfuscates the presentation for computer systems, since those mostly do not understand the data on a semantic level and thus, are not capable to distinguish presentation related data from actual data.

Web scraping or more general screen scraping describes the automated extraction of information from the computer screen or in our case websites. For many applications it is an important pre-processing step before applying machine learning and data analytics (Glez-Peña et al., 2014; Mahto and Singh, 2016).

The usual process of web scraping involves users, that decide what data to extract from websites. The extraction is determined by models given as regex or selector languages like *XPath*. Then in consecutive execution steps scrapers use these models to extract data from websites. This can be done in certain intervals and also throughout a large set of structurally equal websites, that contain different data. If a website changes, for instance if an owner of a website applies A/B testing to compare different websites designs, new models must be fitted. Although, the actual scraping step is executed automatically without user interaction, the model generation is done by a user, who must be an expert or specifically trained if the model fit requires programming.

Besides, scraping large amounts of data from equally structured websites another use case, that motivates our approach comes in the form of browser plugins, web automation (Leshed et al., 2008) or general software, that directly interacts with specific data from websites. If the websites changes a fixed model (as, for instance an XPath expression), that was specified by a programmer might not longer be able to extract the important information. Then a new software

Osterland, T. and Rose, T.
Scoring-based DOM Content Selection with Discrete Periodicity Analysis.
DOI: 10.5220/0011116300003179
In *Proceedings of the 24th International Conference on Enterprise Information Systems (ICEIS 2022) - Volume 2*, pages 280-289
ISBN: 978-989-758-569-2; ISSN: 2184-4992

version is necessary. However, if the model generation is as simple as clicking on an element on the website, we can let the user update the model. In particular, if the model is transferable, such an updated model can be shared among all users of a software and some users might not even notice, that the website changed.

A third use case, that benefits from the proposed selector is supervised machine learning. To train models large data sets needs to be prepared and labeled or categorized. Since the data sets are often large, the combined work of many people is required. In this case, it is an advantage if these people do not need to be experts in programming, that are capable of creating regular expressions or XPath definitions. This enables the utilization of crowd worker platforms, as *Amazon's Mechanical Turk*. Our approach allows the identification of complex selection patterns based on mouse clicks of a user. This makes our approach particularly accessible to untrained personnel.

The general idea of our approach is to weight different paths in the websites DOM tree depending on the user inputs. Thereby, the user inputs are evaluated to obtain a model, that can be used to assess whether a given path is likely to be a selected one. The novelty of our approach is the idea to combine the path scoring with discrete periodicity analysis to identify specific patterns selected by a user. For instance, selecting every second or third row of a table or two consecutive rows followed by one row that is not selected. These are often properties, that can be easily expressed in scripting-based selectors as XPath, but are difficult or elaborate to obtain with click events.

The paper is structured as follows. In Section 2, we review related work, that discusses web scraping and technical approaches to extract specific information. We then start in Section 3 with the introduction of the path scoring approach, that is based on analyses of CSS classes and HTML tag names mapped to states of the DOM tree. We introduce periodicity analysis in Section 3.1 and a bisimulation relation to asses the similarity of subtrees in Section 3.2, which further improves the selection results of our approach. We evaluate the model sizes and the time complexity of our approach in Section 4 and also discuss experimental results. The results of this paper and potential future work are discussed in Section 5.

2 RELATED WORK

Glez-Peña et al. differentiate three different classes of web scraping approaches "(i) libraries for general purpose programming languages, (ii) frameworks and (iii) desktop-based environments" (Glez-Peña et al., 2014). Thereby, the first category supports in the utilization of programming languages as *Perl* or *Python* to extract information from websites. Glez-Peña et al. also count commandline tools, as *curl* or *wget* to this category, what softens the clear separation of the three categories. The second category entails frameworks, as *Scrapy* or *jARVEST* and demarcates itself from the other categories by containing approaches, that are a "more integrative solution" (Glez-Peña et al., 2014). The final category, enables layman programmers to fathom web scraping.

Thereby, the proposed categorization demonstrates the fundamental paradigms: On the one hand approaches, that can be in practice only used by trained personnel, since they require programming and those, that support users with elaborate interfaces. However, the categorization appears partially inconsistent in the sense, that there exists overlaps between categories (some frameworks can be also used as a library (for instance Scrapy)) and we further argue, that tools as *curl* or *wget* are independent programs. It is possible to call these programs within program code, but it stretches the concept of program code libraries.

A different family of approaches to extract information from web pages is based on semantic information. Thereby, DOM elements are enriched with ontological information. The *World Wide Web Consortium* (W3C)[1] proposes a language, called *Web Ontology Language* (OWL), that provides different expressions to define relationsships and affiliations between content. Lamy (Lamy, 2017) presents how this additional information can be used to automatically classify data. Fernández et al. present a semantic web scraping approach based on resource description framework (RDF) models, that are used to map DOM elements to "semantic web resources" (Fernández Villamor et al., 2011). One problem of these approaches is that often semantic annotations are not available for websites.

Mahto et al. discuss in their paper the ethics of web scraping and provide web scraping examples written in *Python* with *BeautifulSoup* (Mahto and Singh, 2016), which is categorized by Glez-Peña et al. as a library (Glez-Peña et al., 2014).

Krosnick and Oney (Krosnick and Oney, 2021) discuss the potential use of scripting based scraping techniques for web automation and end-to-end user interface testing. They conduct two studies to assess usual challenges, that users face when they implement web automation scripts. In the first study all users needed to develop selection scripts in a simple editor,

[1] https://www.w3.org/

while in the second there was a simple IDE that provided visual feedback during development. All test participants were programmers with varying degrees of experience.

Munoz et al. present in their paper μRaptor, which is a tool designed to extract hCard information from webpages (Munoz et al., 2014). They use a training set of DOMs to extract a set of rules in the form of CSS selectors. The rules are then used to identify DOM subtrees, that represent hCards.

The layman category of web scraping tools introduced by Glez-Peña et al. is not primarily targeted by Krosnick and Oney. However, they discuss in their related work tools that use *Programming-by-Demonstration* (PbD) approaches for the model creation. One major motivation of PbD is to empower non-experts to handle complex tasks. Frank (Frank, 1995) demonstrates this for interactive systems by comparing *snapshots* before and after a user interaction. Barman et al. (Barman et al., 2016) use this approach for the selection of DOM elements and similar to our approach they use similarity measures. However, they do not apply periodicity analysis to identify complex selection patterns.

Rousillon developed by Chasins et al. (Chasins et al., 2018) is a tool that uses PbD for web scraping. It allows a user to record actions, that can be replayed for automation. Similar functionality is provided by the academic tools *CoScripter* of Leshed et al. (Leshed et al., 2008) and *SUGILITE* from Li et al. (Li et al., 2017). Additionally, there exists a variety of commercial tools, as *Selenium*[2], *iMacros*[3] and *Cypress Studio*[4].

Kadam and Pakle present *DEUDS* (Kadam and Pakle, 2014) an approach to learn extraction rules from training data. They describe a three step procedure: In the first step a user can interactively select the important information, then the approach will extract *semantic tokens* in the second and hierarchical information in the third step to build a selector model, that allows the page wise extraction of information.

However, most existing approaches, either rely on existing technologies, as CSS selectors or XPath statements or provide their own domain-specific language. In particular for the use case of web automation PbD is used to record user interactions. Our approach employs discrete periodicity analysis to enable the selection of complex patterns. Hence, we target the third category of Glez-Peña et al. that enables layman programmers or web users in general to train models and thus, allow the methodical extraction

[2]https://www.selenium.dev

[3]https://www.imacros.net

[4]https://www.cypress.io/

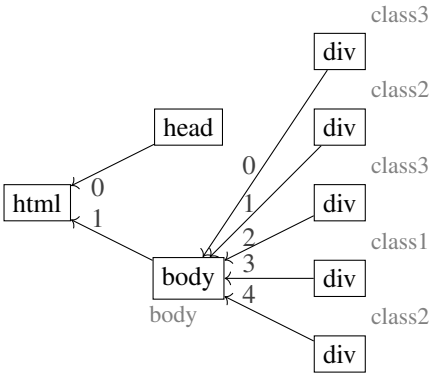

Figure 1: Example of DOM (abstract representation). The CSS classes assigned to the HTML elements are displayed in gray. The index of an element in the child array of the parent, which is important for the periodicity analysis is displayed in dark gray.

of dedicated information from websites. We develop the approach focusing on two applications: For once machine learning requires a lot of carefully reviewed information, that often can only be provided by humans. *Clickworkers* or *crowdworkers* on platforms as for instance *Amazon's Mechanical Turk* are a common solution. However, these workers are rarely programming experts and the quality of their work often depends on the usability of the data processing. The second use case are for instance browser plugins that work with the data of specific websites. Companies and organizations frequently update their websites, for instance when applying A/B testing to find more efficient presentations or in case of regular updates. Often this will break existing selectors specified as regular expressions or *XPath* code. In such cases an update of the browser plugin is necessary, but in particular in case of A/B testing it is not ensured that the developer will obtain the same website version as the plugin user. An adaption of the content selector is difficult. Instead, our approach empowers the user of a browser plugin to retrain the model if she notices, that the existing selection logic is not longer able to extract the important data from a website.

3 PATH SCORING

Figure 1 shows an abstract representation of a document object model (DOM). A DOM structures the HTML (or XHTML) elements of a web page in a tree structure. The root node of a DOM is the `document` that is usually followed by an `HTML` tag. In the example in Figure 1, we start with the `HTML` tag.

In the following we present a DOM as a directed acyclic graph (or *labeled transition system* (LTS))

$\mathbb{G} = (S, \rightarrow, root, TAG, CSS, L)$, with S a set of states, $\rightarrow \subseteq S \times S$ a relation, that represents the child-parent relationship between elements of different levels in the DOM tree, $root \in S$ the root node, which has no *incoming* transitions, i.e., $\{(v, root) \mid \forall v \in S\} \nsubseteq \rightarrow$, TAG is a set of HTML tag names and CSS a set of CSS classes. $L : S \rightarrow 2^{CSS} \cup HTML$ is a labeling function, that assigns CSS classes and HTML tag names to states of the graph. Note, that the mapping L can assign an arbitrary number of CSS classes to a state, but only one HTML tag name. $\mathcal{G}$ is the set of all DOM DAGs.

We further define the set of siblings $siblings(s) = \{s' \mid \exists\, q \in S : (q,s) \in \rightarrow \Rightarrow (q,s') \in \rightarrow\}$. Note, that q is unambiguous due to the tree structure of the DOM. Since, the order of elements in the DOM is important for the presentation, we define the natural total order $<$ on a set of siblings, according to their appearance in the DOM. For example in Figure 1 the order on the set of siblings $siblings(\texttt{BODY}) = \{\texttt{HEAD}, \texttt{BODY}\}$ with $\texttt{HEAD} < \texttt{BODY}$.

We use this graph structure together with the $\rightarrow$ relation to define a path semantic on the DOM. In early experiments, we tested Markov chains for the classification of *selected* elements. Thereby, we modeled different elements of the DOM as states in the Markov chain, that was weighted in accordance to the training data provided by the user, as for instance by selecting some elements. However, a usual DOM contains a substantial number of equal HTML elements and CSS classes. The redundancies within the graph together with the Markov property (memoryless) of the model lead to poor classification results. Better results can be obtained, when we increase the order of the Markov chain. If we increase the order to the path length of the largest training path, we obtain the best classification results, but hardly benefit from the locality of the Markov property anymore.

A path is an ordered sequence of states $\omega = s_1 \cdots s_n$ with $n \in \mathbb{N}$ and $\forall\, i \in \{1, \cdots, n-1\}$ it holds, that $(s_i, s_{i+1}) \in \rightarrow$. The length of a path ω is $|\omega| = n$. We use $|\cdot|$ also to denote the cardinality of sets. The set of all paths over an LTS $\mathbb{G} \in \mathcal{G}$ is denoted as $\mathbb{P}_{\mathbb{G}}$. For two states s, s' in a path we write $s < s'$, if s appears earlier in the path than s'. We call a path ω a *root* path if $s_1 = root$ and a *leaf* path, if $\nexists\, x \in S : (s_n, x) \in \rightarrow$. A path, that is a *root* path *and* a *leaf* path is called *complete*. In Figure 1 paths, that contain the `div` tags are *leaf* paths and a non-rigorously written example for a complete path is $\omega = \texttt{html body div}$.

The *training set* T is a set of *root* paths. In practice, we can obtain this set by adding a click event listener to a website and then construct the path by iterating through the parents of the selected element until we reach the document root. Our model (see Equation 2) evaluates the sizes of training paths and thus, we are not limited to leaf paths in the training set. However, it is often desired, since the goal is the extraction of information, which is often the text attribute of a leaf HTML element. Note, that when we use the click event listener approach to obtain the training samples it is easy for a user to hit the wrong element. So for instance instead of hitting a `span` element, that directly contains the desired information the surrounding `div` element is chosen. This is a challenge that must be considered, when collecting the training samples with this approach.

In the following, we define the set of training tags $T_{TAG} = \{L(s) \cap TAG \mid \forall\, s \in \omega\, \forall\, \omega \in T\}$ and the set of training CSS classes $T_{CSS} = \{L(s) \cap CSS \mid \forall\, s \in \omega\, \forall\, \omega \in T\}$. This allows us to define the scoring functions, that measures the overlap of TAGs and CSS classes in the training set and a given path ω:

$$p_Q(\omega) = \frac{|\{s \mid (\{L(s)\} \cap Q \neq \emptyset \wedge s \in \omega\}|}{|\omega|} \quad (1)$$

In the equation $Q \in \{T_{CSS}, T_{TAG}\}$, i.e., we use the same scoring function to measure the overlap in CSS classes and HTML tags. Note, that in case of perfect overlap, that means in case a path $\omega \in \mathbb{P}$ is part of the training set $p_Q(\omega) = 1$. Generally it holds, that $p_Q(\omega) \in [0,1], \forall\, \omega \in \mathbb{P}$.

We gradually penalize deviations from path lengths, that are present in the training set.

$$p_k(\omega) = 1 - \frac{|k - |\omega||}{max(k, |\omega|)} \quad (2)$$

We determine k by evaluating the expression $\min_{k=|\omega_t|, \omega_t \in T} ||\omega| - k|$. That means, k is the length of a path in the training set, that is closest in length to the path ω. Again $p_k(\omega) \in [0,1], \forall\, \omega \in \mathbb{P}$ and $p_k(\omega) = 1$ if the scored path $\omega \in \mathbb{P}$ is present in the training set.

3.1 Discrete Periodicity Analysis

The most distinctive feature of our approach compared to other approaches is based on the utilization of discrete periodicity analysis to make assumptions about the intended selection pattern of a user. For example often data on websites is provided as tables. If only every second row or even a more complex pattern, as two consecutive rows followed by three irrelevant rows need to be selected most mouse-based approaches cannot provide this functionality.

Scripting based approaches, as *XPath* or *CSS selectors* have an advantage in these situations, but are not equally accessible by uneducated workers. Of

course, the approach is not limited to table rows, but can be also used to extract patterns from arbitrary HTML elements, as for instance float or CSS grid screen layouts.

The discrete periodicity analysis, that we employ originates in approaches to mine association rules (Agrawal and Srikant, 1994), which are probably best known for the association example of diapers and beer, which is best placed next to each other in stores.

Agrawal and Srikant present in their paper (Agrawal and Srikant, 1994) the *A priori property*, which uses subsets of potential patterns to analyze the frequency of occurrence of larger patterns. This can then be used to derive a confidence score, that a certain pattern appears periodically. The challenge during the mining process is that, we assume noisy data, which means that the pattern will not match exactly at certain periods.

Hang et al. (Han et al., 1999) define the problem of periodicity mining as follows: For a time series of features $S = D_1, D_2, \cdots, D_n$, where $D_i, i \in \{1, \cdots, n\}$ are sets generated from a sequence of chronologically ordered database entries, we try to mine *patterns* $s = s_1 \cdots, s_p$ with $s_i \in L, i \in \{1, \cdots, p\}$ with $L = ((D_1 \cup \cdots \cup D_n) - \emptyset) \cup \{*\}$. Hang et al. use $*$ as a *don't care* character. If this appears in a pattern it indicates, that we are not interested in the character at this position, but only on the periodicity of the surrounding elements.

To evaluate the quality of an extracted pattern, they define a confidence score, that is based on the occurrence frequency of a pattern s:

$$frequency\_count(s) = |\{i \mid 0 \leq i \leq m, s \text{ holds in } D_{i|s|+1} \cdots D_{i|s|+|s|}\}| \quad (3)$$

The confidence is then the frequency divided by m, which is the number of times the pattern fits into the analyzed sequence.

$$conf(s) = \frac{frequency\_count(s)}{m} \quad (4)$$

Note that we can control the minimal required number of pattern repetitions with the confidence value. If for instance a pattern needs to be at least two times present in the sequence, we can derive a minimal bound and only work with patterns that have at least this confidence. With regard to our content selection use case, we use this minimal bound (of two present repetitions), since one occurrence cannot be identified as periodic and more than two are too much work for a user to train. Although, the confidence will increase if the user further selects elements, that fit the pattern. For instance our model must respond, if a user selects two alternating rows in a table.

However, not every selection of a user constitutes a pattern repetition. Depending on the pattern multiple selections represent one pattern and therefore, more selections from the users are necessary to reach the confidence level of at least two. For instance in Figure 1, if the user wants to select the first and the second `div`, but not the third `div`, she needs to click on the first, the second, the fourth and the fifth `div` to reach the required confidence.

Hang et al. present in their paper (Han et al., 1999) a more efficient adaption of the association rule mining algorithm presented by Agrawal and Srikant, which they call *max-subpattern hit-set*. They introduce a sophisticated tree datastructure, the *max-subpattern tree*, which efficiently organizes scanned patterns and their variations. In a consecutive step we can evaluate the datastructure, to derive the patterns with the highest confidence. For an explanation of the algorithm, we refer the reader to the paper of Hang et al. (Han et al., 1999).

In the following, we refer to the *max-subpattern hit-set* algorithm as a function $max\_hit\_set : L^{\mathbb{N}} \times \mathbb{R} \times \mathbb{N} \rightarrow 2^{L^{\mathbb{N}} \times \mathbb{N}}$, that maps from a sequence ($L^{\mathbb{N}}$) a minimal confidence threshold ($\mathbb{R}$) and a pattern length ($\mathbb{N}$) to a set of patterns ($L^{\mathbb{N}}$) and the number of their occurrences ($\mathbb{N}$). We further use the restricted feature set $L = \{1, *\}$. Equation 5 shows, how we encode the elements of a training sample as a sequence $L^{\mathbb{N}}$, that we can feed into the *max-subpattern hit-set* algorithm.

$$pattern : s \mapsto (x_{s^1}, \cdots, x_{s^m}), with \quad \begin{cases} x_{s^i} = 1 & \exists\, \omega' \in T : s^i \in \omega' \\ x_{s^i} = * & otherwise \end{cases} \quad (5)$$

with $s^i \in siblings(s)$, $s^1 < \cdots < s^m$ and $1 \leq i \leq |siblings(s)| = m \in \mathbb{N}$. We use superscript indices to indicate, that we iterate over the siblings of a state and *not* over the states of a path.

The idea of Equation 5 is to encode DOM elements of one level, that are part of the training set as 1 and all the remaining elements as $*$ (don't care). For an element we look at all its siblings, whether it is also part of the training set. For example if in Figure 1 a user wants to select every other `DIV` element and selects the first and the third `DIV` element. Then we construct an input sequence $1 * 1 * *$ and the pattern $1*$ occurs two times in the sequence.

The pseudocode in Algorithm 1 describes how we interface the periodicity algorithm. For a given state s of a training path $s \in \omega$, we derive the input pattern with the function of Equation 5. For sequences of length $|p| = 1$, we directly return the considered state s if it is part of the training set, since there is no sense

Input: $s \in \omega$
Output: $period(s) = R$
$p \leftarrow pattern(s)$;
if $|p| = 1$ **then**
  if $\exists\, \omega' \in T : s \in \omega'$ **then**
    return $\{s\}$;
  else
    return $\emptyset$;
  end
end
$U \leftarrow \emptyset$;
while $i \in \{2, \cdots, |p|\}$ **do**
  $U \leftarrow U \cup \{max\_hit\_set(p, 2i/|p|, i)\}$;
end
$M \leftarrow \{(p,q) \mid (p,q) \in U : \max|p| \wedge \max q\}$;
if $|M| > 1$ **then**
  // request additional training data
else
  $r \leftarrow M[0]$;
end
$R \leftarrow= \{s' \mid s' \in siblings(s) : p'[< (s')$
 $\bmod |p'|] = 1 \vee p[< (s')$
 $\bmod |p|] = 1, (p', q) = r\}$;

Algorithm 1: Identify the periods.

analyzing the periodicity in sequences of length one and if it is part of the training set it is translated into a 1 in the input pattern. For sequences of length $|p| > 1$ we apply the *max_hit_set* function for every period length $2 \leq i \leq |p|$. From the unified result set, we choose the detected pattern, that is largest and has the highest frequency of occurrences. If the set M of selected patterns contains more than one element we require additional training data, otherwise we will choose the pattern. We use this selected pattern to derive those siblings, that match the extrapolated pattern. Note, that we use $< (s) = i$ as a loose syntactic expression to denote the index of s in the order of its siblings $s_1 < \cdots < s_{i-1} < s < s_{i+1} < \cdots < s_l$ and $p[i]$ to access the i'th element of the pattern. We also select an element if it is one in the input sequence, i.e., we force the inclusion of states, that are part of the training set, since those are provided by the user and even if they do not belong to a pattern the user wants those elements to be selected.

We use Algorithm 1 to score paths as follows:

$$p_p(\omega) = \frac{|\{s \mid s \in \omega \wedge s \in period(s)\}|}{|\omega|} \qquad (6)$$

Every state of a path $s \in \omega$ is checked, whether it is part of a detected periodic pattern. We count the number of states at which this is the case and divide it by the number of total states in the path. For training samples the score will be $p_p(\omega) = 1, \omega \in T$ and $p_p(\omega) \in [0,1], \forall\, \omega \in \mathbb{P}$.

We combine the introduced individual scoring functions into a combined score in Equation 7 and further introduce parameters $a_{TAG}, a_{CSS}, a_k, a_p$ with $a_{TAG} + a_{CSS} + a_k + a_p = 1$ to weight certain scores differently if required.

$$p_{path}(\omega) = \frac{1}{4}(a_{TAG} p_{T_{TAG}}(\omega) + a_{CSS} p_{T_{CSS}}(\omega) + a_k p_k(\omega) + a_p p_p(\omega)) \qquad (7)$$

An example of a situation in which different weights become useful is if a websites colors alternating rows of a table differently by assigning different CSS classes. In this case we can reduce the weight of the CSS scoring function to make sure, that it does not weight the rows of the table too differently.

3.2 (Bi)Simulation to Identify Similar Subtrees

For the example presented in Figure 1, the periodicity analysis will work in the sense, that it selects those `div` elements, that are part of the identified pattern. However, this will not directly work if the `div` element is again the root element of a subtree. This is demonstrated in the example in Figure 2.

If a user selects sub-elements of the first and the third `div` the fifth `div` will be identified by the periodicity analysis, but this will not influence the scores of the leaf elements in the subtree. That means, we cannot decide whether to select the first or the second `span`.

We use a bisimulation relation to compare the branching structure of different subtrees in the DOM. Baier and Katoen present a definition of bisimulation equivalence on labeled transition systems (Baier and Katoen, 2008), which we use in a slightly adapted form.

For two labeled transition systems $\mathbb{G}_i = (S_i, \rightarrow_i, root_i, TAG_i, CSS_i, L_i), i \in \{1,2\}$ a bisimulation for $(\mathbb{G}_1, \mathbb{G}_2)$ is a binary relation $\mathcal{R} \subseteq S_1 \times S_2$ with:

1. For $s \in S_1 \wedge s = root_1 : \exists\, s' \in S_2 \wedge s' = root_2 : (s,s') \in \mathcal{R}$ and for $s \in S_2 \wedge s = root_2 : \exists\, s' \in S_1 \wedge s' = root_1 : (s,s') \in \mathcal{R}$
2. $\forall\, (s_1, s_2) \in \mathcal{R}$:
   (a) $L_1(s_1) = L_2(s_2)$
   (b) $\forall s \in S_1$ with $(s_1, s) \in \rightarrow_1$: $\exists\, s' \in S_2$ with $(s_2, s') \in \rightarrow_2$ and $(s,s') \in \mathcal{R}$
   (c) $\forall s \in S_2$ with $(s_2, s) \in \rightarrow_2$: $\exists\, s' \in S_1$ with $(s_1, s') \in \rightarrow_1$ and $(s,s') \in \mathcal{R}$

We write $\mathbb{G}_1 \sim \mathbb{G}_2$ for two labeled transition systems $\mathbb{G}_1, \mathbb{G}_2$, that are bisimilar.

Note, that we deviate from the definition of Baier and Katoen in particular on the first item, since transition systems are usually not limited to directed acyclic graphs (DAGs). Hence, there exists a set of initial states. Since our transition systems represents a DOM tree or a subset of it, we always have the case, that the initial set is a single state, that we designate as *root*.

The first item of the second property $2-1: L_1(s_1) = L_2(s_2)$ allows us to decide the equality of states. We defined the labeling function L based on the *TAG* and assigned *CSS* classes. States that share these properties are considered as equal by our algorithm. The second and the third property enforce the same branching structure on the tree.

We write $\mathbb{G}_{sub}(s) \subseteq \mathbb{G} = (S, \rightarrow, root, TAG, CSS, L) : s \in S$ for a subtree induced by choosing a state $s \in S$ as its root and only keep states and transitions, that are connected in a successor relationship. $\mathbb{G}_{sub} : \mathcal{G} \times S \rightarrow \mathcal{G}$ with $(\mathbb{G}, s) \mapsto (S', \rightarrow', s, TAG', CSS', L')$:

- $S' := S \cap \{s' \mid \exists\, \omega \in \mathbb{P}_{\mathbb{G}} : s \in \omega \wedge s' \in \omega \wedge s < s'\}$
- $\rightarrow' := \{(s, s') \mid (s, s') \in \rightarrow \wedge\, s, s' \in S'\}$
- $TAG' := TAG \cap \{q \mid \exists\, s' \in S' : L(s') = (A, q), A \in 2^{CSS}\}$
- $CSS' := CSS \cap \{x \mid \exists\, s' \in S', \exists\, A \in 2^{CSS} : L(s') = (A, q) \wedge x \in A, q \in TAG\}$
- $L' := L|_{S'}$

$L|_{S'}$ denotes the restriction of the labeling function to the preimage set, that only contains states of the subtree. Consequently, we obtain a subtree, that only comprises the restricted labeling function with corresponding CSS classes and tags that are still present in the remaining states.

For a path ω and a state $s \in \omega = (s_1, \cdots, s_{j-1}, s, s_{j+1}, \cdots, s_l)$ and $\nexists\, \omega' \in T : s \in \omega'$, i.e., the state s is not part of a training path and $s \in period(s)$ is part of an identified pattern we search for bisimilar subtrees induced by the siblings of s, that are part of a training path and construct a path ω'' that is used in the scoring function $p_p(\omega'')$ with the following method:

$$\begin{aligned} &\forall s' \in siblings(s) : (s' \neq s \wedge \\ &\exists\, \omega' = (s'_1, \cdots, s'_{i-1}, s', s'_{i+1}, \cdots, s'_k) \in T : s' \in \omega' \wedge \\ &\quad s' \in pattern(s) \wedge \mathbb{G}_{sub}(s) \sim \mathbb{G}_{sub}(s')) \Rightarrow \\ &\qquad (s_1, \cdots, s_{j-1}, s, s'_{i+1}, \cdots, s'_k) = \omega'' \end{aligned} \quad (8)$$

Equation 8 describes our method to apply the periodicity scoring on paths selected by the periodicity analysis. For selected paths we search in the training set for a path that is most similar to the selected one and then combine the periodicity data from these two paths by joining the state sequences. This allows us to transfer the periodicity analysis model from the training set to paths, that are not directly part of the training set, but are identified by the periodicity analysis.

Of course, if the training set is large there are multiple states s' in different training paths, that might fit the target state. However, since all these eligible state induce bisimilar subtrees, we can select an arbitrary training state and obtain the same result.

Note, that the bisimulation relation is strict, since it requests that transition steps can be replayed in both systems, i.e., a specific move in one system must be repeatable in the other and vicit versa. For our approach it is often sufficient if we can find a training path, that is capable of repeating every transition step of the target system. That means in contrast to demanding an equivalence relationship, we just demand it in one direction. This can be achieved with the help of a *simulation* relation, which is basically a one-sided *bisimulation* relation.

4 EVALUATION

For the evaluation of our approach we analyze the computational complexity of the scoring and also the space, that is required to store the models derived from the training data. This is important, since those will be stored for the extraction of specific content and it might be necessary to update this model if a webpage changes. In case of an application, as for instance a browser plugin, the model must be shared among all the users of the application.

We are also interested in the performance of our scoring approach with respect to accuracy and precision. That means, we measure the elements that the approach extracts correctly and those, that it extracts, but are not intended, i.e., true positives and false positives.

For this we conduct an experiment. Our methodology is to randomly select leaf DOM elements, that contain text content, i.e., ASCII characters, from the seven most popular websites *google.com*, *youtube.com*, *facebook.com*, *wikipedia.org*, *yahoo.co.jp*, *amazon.com*, *instagram.com* as listed by *statista.com*[5]. These randomly selected elements simulate the user input and thus our training data. We derive models from these elements and then in a second step evaluate the scores of the paths on the

[5]https://www.statista.com/statistics/1201880/most-visited-websites-worldwide/

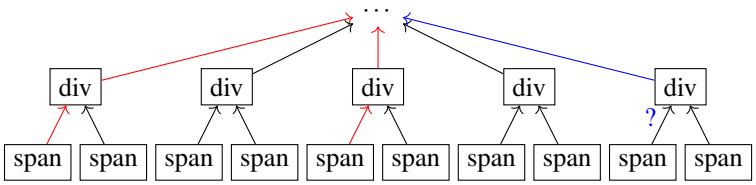

Figure 2: Example to showcase the necessity of evaluating the bisimilarity of subtrees in case of identified periodicity. The red lines represent the training paths and the blue path is selected by the periodicity analysis. The question mark indicates the problem, that a selection path for the subtree cannot be directly deduced from the periodicity analysis of the parent elements.

DOM for the model. We measure execution time and the correctness of selected elements.

4.1 Model Size and Runtime Complexity

The model size depends directly on the training set. For a path $\omega \in T$ we need to store for every state $s \in \omega$ the corresponding tag TAG_s, the assigned CSS classes CSS_s and a pair of $(p,q) \in \{0,1\}^{\mathbb{N}} \times \mathbb{N}$, that contains the evaluated pattern and its period.

In worst case every path of a DOM is a path in the training set, and we use the number of states in the DOM $|S|$ as an upper bound for the pattern. Thus, we have a complexity of $O(|S||CSS| + |S| + |S|^2)$, where the first term assumes, that every available CSS class is assigned to every state in the DOM. The second term presents, that every state is unambiguously mapped to a tag and the third term assumes, that we need to store for every state a pattern consisting of all the other states in the DOM. Of course, in practice the worst case model size is not realistically obtained. CSS classes are usually assigned to only a very small number of states and the number of siblings of most states are by a large margin smaller than the total number of states in the DOM. In Figure 3, we analyzed the number of elements in the DOM, the average CSS classes assigned to a state and the average number of siblings of a state collected from the popular websites mentioned above.

Note in particular, that the average number of classes and siblings are much smaller than the total number of DOM elements. Also, the number of CSS classes assigned to a single DOM element is below 20. Hence, in practice we can expect much smaller models, than suggested by the worst case analysis.

To determine the time complexity of the proposed approach we need to distinguish the evaluation phase and the training phase. For the evaluation we only need to traverse the DOM tree, what can be done with a tree traversal algorithm in $O(|S|)$ with S the states in the tree. During traversal, we iteratively extend the paths and apply the scoring. The counts of the earlier scoring can be cached, such that we are not forced to

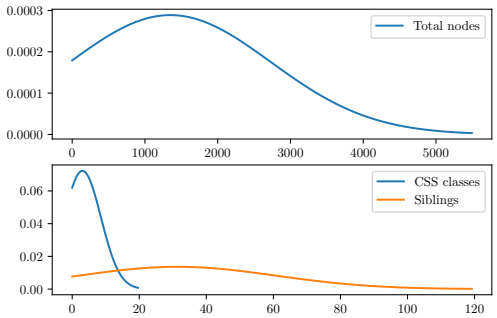

Figure 3: Total number of DOM elements, average number of classes assigned to DOM states and the average number of siblings collected from the most popular websites.

traverse the entire path for every additional state with which the path is extended. We can use the index of a state to determine, whether it is part of a pattern to evaluate the periodicity score.

The time complexity will deteriorate if we use bisimulation to compare subtrees of identified patterns. Baier and Katoen give the complexity of checking the equivalence of two labeled transition systems LTS_1, LTS_2 as $O((|S_1| + |S_2|) * |AP| + (M_1 + M_2)\log(|S_1| + |S_2|)$ (Baier and Katoen, 2008). S_1, S_2 represent the set of states, AP the sets of labels, which are in our case composed by CSS and TAG and M_1, M_2 is the number of edges in the corresponding transition systems.

The training phase requires in worst case $O(|S|^3)$ operations. This is since in worst case every path and thus, every state in the DOM is part of the training set. Jiawei et al. (Han et al., 1999) specify the runtime complexity of the *max-subpattern hit-set* algorithm with $O(M)$, where M is the length of the sequence, that need to be scanned for detecting patterns. We call the *max-subpattern hit-set* algorithm for every periodicity $p \in \{1, \cdots, M\}$, which means that the complexity will be $O(M^2)$. Since, the input sequence is generated by the siblings of a state, and S is the set of all states, we can use $|S| = M$ as an upper bound. With the necessary traversal through the DOM tree, i.e., the training set (remember in worst case every path in the DOM is a training path) we obtain the complexity of $O(|S|^3)$.

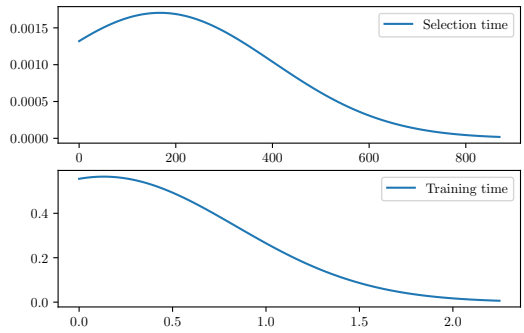

Figure 4: Total number of DOM elements, average number of classes assigned to DOM states and the average number of siblings collected from the most popular websites.

Since, we randomly select a single leaf path for training in our experiment, we do not need to identify selection patterns, which is difficult to test automatically. The approach identified in 100% of the test cases the correct DOM element, which is explained by the periodicity analysis, that favors element indices, that are part of the training set. That means, even if there is only a single training path and thus, it is not possible to derive a sequence for periodicity analysis, that contains more than one 1, this single one is stronger weighted than all the siblings of the element.

Figure 4 particularly emphasizes the expensive costs of the periodicity analysis of $O(|S|^3)$. In our implementation, we applied the periodicity analysis during the selection phase. Of course, in a production implementation this should be done during training and then the results of the analysis must be stored in the model, as described above. The numbers of the abscissa represent time in milliseconds. For websites with large DOMs trees the selection training required almost a full second, although the majority requires around 200 milliseconds. Please note, that we focused in our implementation on experimenting and correctness and less on computational complexity. Hence, there is much room for improvement. Both, with regard to efficient execution and to compact storing of trained models.

5 CONCLUSIONS

In this paper we demonstrated how periodicity analysis can be used as an approach to empower non-expert users to extract specific information from websites. Our approach represents an extension of existing approaches, since it allows the simple definition of complex selection patterns derived from user clicks. We further employed bisimulation relations to measure the similarity of subtrees to further improve our approach.

Although, our complexity analysis shows, that the computational complexity and model sizes of our approach can be substantial in worst case, experimental results and empirical evaluation of the DOM trees of the most popular websites revealed, that in practice the worst case considerations are rarely observed and thus, the performance is much better in practice. Additionally, we need to distinguish the runtime complexity of the training phase and the selection phase. The majority of required computations is part of the training phase, which is for most uses cases less frequently necessary, than the less expensive selection phase. The selection phase can be efficiently computed in $O(|S|)$, but worsens when we employ bisimulation to check different subtress.

The experiment also indicates possible future work, as the performance tuning of our algorithm. A termination condition in the selection phase will prevent the traversal of the entire DOM tree and we can utilize for periodicity analysis the fact, that in contrast to our worst case assumption, that every DOM path is a training path, the training set is usually much smaller.

ACKNOWLEDGEMENTS

This research was partially founded by the *b-it foundation* (Bonn-Aachen International Center for Information Technology)[6].

REFERENCES

Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, page 487–499, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Baier, C. and Katoen, J. (2008). *Principles of Model Checking*. MIT Press.

Barman, S., Chasins, S., Bodik, R., and Gulwani, S. (2016). Ringer: web automation by demonstration. In *Proceedings of the 2016 ACM SIGPLAN international conference on object-oriented programming, systems, languages, and applications*, pages 748–764.

Chasins, S. E., Mueller, M., and Bodik, R. (2018). Rousillon: Scraping distributed hierarchical web data. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*, pages 963–975.

[6]https://www.b-it-center.de/

Fernández Villamor, J. I., Blasco Garcia, J., Iglesias Fernandez, C. A., and Garijo Ayestaran, M. (2011). A semantic scraping model for web resources-applying linked data to web page screen scraping.

Frank, M. (1995). Inference bear: Designing interactive interfaces through before and after snapshots.

Glez-Peña, D., Lourenço, A., López-Fernández, H., Reboiro-Jato, M., and Riverola, F. F. (2014). Web scraping technologies in an api world. *Briefings in bioinformatics*, 15 5:788–97.

Han, J., Dong, G., and Yin, Y. (1999). Efficient mining of partial periodic patterns in time series database. pages 106–115.

Kadam, V. B. and Pakle, G. K. (2014). Deuds: Data extraction using dom tree and selectors. *International Journal of Computer Science and Information Technologies*, 5(2):1403–1410.

Krosnick, R. and Oney, S. (2021). Understanding the challenges and needs of programmers writing web automation scripts. In *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 1–9.

Lamy, J.-B. (2017). Owlready: Ontology-oriented programming in python with automatic classification and high level constructs for biomedical ontologies. *Artificial intelligence in medicine*, 80:11–28.

Leshed, G., Haber, E., and Matthews, T. (2008). Coscripter: Automating & sharing how-to knowledge in the enterprise. pages 1719–1728.

Li, T., Azaria, A., and Myers, B. (2017). Sugilite: Creating multimodal smartphone automation by demonstration.

Mahto, D. K. and Singh, L. (2016). A dive into web scraper world. In *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 689–693.

Munoz, E., Costabello, L., and Vandenbussche, P.-Y. (2014). μraptor: A dom-based system with appetite for hcard elements. In *LD4IE@ ISWC*, pages 67–71. Citeseer.