Stable web scraping: an approach based on neighbour zone and path similarity of page elements

Peng Gao*
Laboratory of Information Security and Software Engineering, NARI Group Corporation/State Grid Electric Power Research Institute, No. 8, Nanrui Road, Gulou District, 210003, Nanjing, Jiangsu, China Email: gao.itslab@gmail.com *Corresponding author

Hao Han
Konica Minolta, Tokyo, Japan Email: han@computer.org

Junxia Guo
College of Information Science and Technology, Beijing University of Chemical Technology, 4# 15 Beisanhuan East Road, Chaoyang District, Beijing, 100029, China Email: gjxia@mail.buct.edu.cn

Motoshi Saeki
School of Computing, Tokyo Institute of Technology, Ookayama 2-12-1-W8-71, Meguro, Tokyo 152-8552, Japan Email: saeki@se.cs.titech.ac.jp

Abstract: Web scraping techniques based on XPath enable users to consistently extract information of interest from webpages that do not provide a structured interface. However, XPath-based extraction is likely to fail when encountering page variants, resulting in a high cost of repair. Countermeasures based on pattern matching or model learning often require careful pre-processing, which is not suitable for cases where the target data is frequently re-designated. In this paper, we present a new extraction method for the stable scraping of arbitrary designated data from webpages. Instead of attempting to find the desired data directly, we first determine its approximate location in the changed page, called the neighbour zone. Then we search for
the precise location by ranking the path similarity of page elements within the neighbour zone. Experiments on a large set of real-world webpages show that our method has better stability for web scraping, compared with the XPath-based extraction. In the two datasets, 0.118 and 0.891 F1-score were increased respectively.

**Keywords:** webpage; web scraping; semi-structured data extraction; XPath expression; stability; HTML tree; node distance; path similarity.


**Biographical notes:** Peng Gao received his BEng in Electronic and Information Engineering from the Nanjing University of Technology in 2005. He received his MEng in Informatics from the Kyushu University, Japan in 2011. He is currently a Research Engineer in the Laboratory of Information Security and Software Engineering, NARI Group Corporation/State Grid Electric Power Research Institute, China. His current research interests include web information extraction, integration and web security.

Hao Han received his PhD in Computer Science from the Tokyo Institute of Technology in 2009. He is currently a R&D Engineer at Konica Minolta, Japan. His research interests focus on web engineering. He is a member of IEEE CS, ACM, JSSST, DBSJ, IEICE and IPSJ.

Junxia Guo is currently a Lecturer at College of Information Science and Technology, Beijing University of Chemical Technology, China. Her current research interests include web information extraction, web service, mashup and web mining. She received her PhD in Computer Science from the Tokyo Institute of Technology, Japan, MSc from the School of Mechanical Electronic and Information Engineering, China University of Mining and Technology, Beijing, China and BSc from the College of Information Engineering, Taiyuan University of Technology, China.

Motoshi Saeki received his BEng in Electrical and Electronic Engineering, MEng and DrEng in Computer Science from the Tokyo Institute of Technology in 1978, 1980, and 1983, respectively. He is currently a Professor of Computer Science at Tokyo Institute of Technology. His research interests include requirements engineering, software design methods, software process modelling, and computer supported cooperative work (CSCW).

1 **Introduction**

Partially extracting the designated data from the web is a prerequisite for further web data analysis, management and application. Web scraping is also known as web data extraction, web harvesting or wrapper, which is a process of automatically extracting the required data from publicly available webpages. For instance, comparison shopping
Stable web scraping

sites like Google Shopping (https://www.google.com/shopping) extract products’ prices, description and rating from different retailers for intelligent recommendation.

Consistently extracting such information is not a trivial task because most of them are presented in a semi-structured manner, typically in HTML format with no structured interface. The common solution is to parse the page into a tree representation and evaluate an XPath (XML path language) (Robie et al., 2014) on it, which specifies how to traverse the trees. The solid line in Figure 1 shows an HTML tree of a product page simplified from a shopping site. To extract the price value carried by the text node with label 6, we can use the following XPath expressions, where \( P_1 \) and \( P_2 \) are an absolute and a relative path respectively.

\[
\begin{align*}
P_1 &= /html/body/div[2]/div[2]/table/tr[1]/td[2]/text() \\
P_2 &= //div[@id='price']/*/tr[1]/td[2]/text()
\end{align*}
\]

Because modern websites often use server-side scripts to generate such pages, e.g., other categories of electronic products, may well share the same template, reusing XPaths becomes a powerful method to extract information of interest for a broad range of applications.

Figure 1 Sample HTML trees of a webpage and a page variant (see online version for colours)

Notes: A webpage and its page variant are mixed in Figure 1. The page variant just inserts thick dashed branch marked as inserted in the page. Other nodes are unchanged.

However, the scraping process based on XPath suffers from the stability problem which makes the extraction fail frequently. On the one hand, websites use similar but inconsistent templates to construct even pages of the same category. Han and Tokuda (2008) showed that the top page of Yahoo! News (https://www.yahoo.com/news/) and BBC Country Profiles (http://news.bbc.co.uk/2/hi/country_profiles/default.stm) used three similar templates for the same page at one time. On the other hand, during a period of time, the inner structure of a webpage may change at any time without notification because the layout is updated. According to Adar et al. (2009), 41.6% of 55,000 HTML
pages changed every hour. We refer to such a webpage as a *page variant* where the initial XPath fails to repeatedly extract the designated information.

As illustrated in Figure 2, the stability problem caused by a page variant can be defined as follows. Let $T_1$, $T_2$ denote HTML trees of a webpage $w_1$ and its page variant $w_2$; $X'$ the leaf node of $T_2$ namely *goal* which corresponds to *target* $X$ in $T_1$ that carries the desired data designated by users; $P(X)$, $P(X')$ the XPath of $X$, $X'$ respectively.

- We apply $P(X)$ in $T_2$, which does not return $X'$.

From another point of view, let $P(X), P(X')$ be absolute XPaths, then if $P(X) \neq P(X')$, the extraction fails. For example, as shown by the thick dashed line in Figure 1, when a new td element carrying an image node with label *inserted* is inserted, the extraction for the price using XPath $P_1$ or $P_2$ fails, which returns nothing in this case.

**Figure 2** Problem statement (see online version for colours)

![Diagram](image)

To address this problem, many research efforts have been presented. However, only a few of them are suitable for application scenario: a customisable web data scraper. The target data may frequently be re-designated by users. For example, different data from multiple webpages are obtained by an end-user mashup tool (Guo and Han, 2014) to be immediately presented to users. Therefore, we aim to make the scraper without pre-processing steps and as stable as possible. The specific challenges are that with no help from:

1. a pre-processing like model learning or pattern designing
2. visual supplementary files like cascading style sheets (CSS), searching the designated data automatically is difficult.

We discuss these challenging issues in Section 6.5.3 and analyse related techniques in Section 7.

Accordingly, we wonder why human can accurately locate the required information without any preparation when confronting such a problem. By intuitively observing the layout and HTML tree structure of webpages, we find that the high similarity of the following features of *target* in $T_1$ and *goal* in $T_2$ helps people to identify it correctly.

- **Layout relationship** (e.g., a distance between locations, neighbour information)
- **Element characteristic** (e.g., HTML element type, semantic information)
Inspired by these observations, in this paper, we present a new approach. Instead of exploring the goal directly, we first use the layout relationship to determine its approximate location, namely neighbour zone, which consists of a fragment of the leaf nodes. Then we search the most likely node within the neighbour zone as the precise location of goal by ranking the path similarity based on characteristics derived from their XPath.

The main contribution of our work is that we designed and implemented a highly stable extraction method that supports arbitrary target data selection in a page. It only uses HTML source files and needs no pre-processing, which makes it be compatible with existing XPath-based methods and flexible for customised extraction. Experiments on two complementary datasets, containing more than 30,000 target data selected from 102 real-world websites, show that our approach’s advantage over the XPath is in terms of extraction stability. The originality of our approach is as follows.

- It develops a distance measurement between a node and a leaf node in the HTML tree.
- It proposes a method to locate the possible location of the target node in the HTML tree of a page variant.
- It presents a method to evaluate XPath similarity from edit, semantic and structure aspects.

The rest of this paper is organised as follows. An overview of the approach is given in Section 2. The concepts of data type and node distance, which depict the element characteristic and layout relationship, are introduced in Section 3. The method for acquiring the neighbour zone is addressed in Section 4. The method to measure similarity of page elements is presented in Section 5. Section 6 shows the experiments we conducted and the evaluations to show the effectiveness of our proposal. Section 7 discusses related work and Section 8 concludes the paper.

2 Overview of our approach

Some notations used in the paper are as follows. Let $T_1$ and $T_2$ denote rooted ordered labeled trees parsed from the webpage $w_1$ and a page variant $w_2$ respectively. Suppose we have a continuous ordering for each tree $T$, and then $T[i]$ means the $i^{th}$ node of $T$ in the given ordering. If the context is clear, we use the ordering $i$ as an abbreviation to denote $T[i]$. The label of each node in $T$ is its HTML node name. When we mention a comparison of two nodes, we mean to compare their labels. In order to clearly show the correspondence between the nodes in $T_1$ and the nodes in $T_2$ without relabelling the node order in $T_2$, for example, we write them as $®$ (target) and $®$ (goal) respectively.

The main components and workflow of our approach are depicted in Figure 3. Because directly locating goal in $T_2$ is difficult, we start from the observations. For the layout relationship mentioned in Section 1, we are concerned with the following two aspects (acquire neighbour zone in Figure 3).
There are some unchanged nodes appearing in both the webpage and its page variant. We refer to them as the unchanged node pair set \( U \) which is a set of node pairs between \( T_1 \) and \( T_2 \) satisfying \( U = \{<i, j>| \text{node } T_1[i] \text{ and node } T_2[j] \text{ are matched (defined in Section 4.1)}.\} \).

There are some page elements in a webpage, and the relative positions between two of them remain unchanged or slightly changed in page variants. The position between two elements in the HTML tree is measured by node distance which is a set that consists of distances’ values from different dimensions.

Figure 3  Workflow of our stable extraction approach based on neighbour zone and path similarity

In the following, we denote an unchanged node pair \(<i, j>\) in the set \( U \) as \( U_{<i, j>} \). We use \( U_{<i>} \) and \( U_{<j>} \) to denote \( T_1_{<i>} \) and \( T_2_{<j>} \) respectively. We calculate the node distance between each \( U_{<i>} \) and \( \text{target} \) in \( T_1 \) and then list up leaf nodes having the same distance in \( T_2 \) to getting the node(s) namely centre node candidate(s). The centre node determines the possible location (neighbour zone) of \( \text{goal} \).

For the element characteristic mentioned in Section 1, we mainly use it to search \( \text{goal} \) within the neighbour zone. The search is based on the following observation.

- The element characteristic of \( \text{goal} \) is more similar to \( \text{target} \) than other nodes within the neighbour zone.

We measure such similarity based on characteristics derived from the XPath, including property, tag path, attribute path, affiliation and list order (compute path similarity in Figure 3). Finally, we select the one from the neighbour zone with the highest score as \( \text{Goal} \). In Sections 3, 4 and 5, we give the details of each component.
3 Data type and node distance of page elements

3.1 Data type

Webpages display information through a variety of media such as text string, images, videos, etc. Such visible contents are carried by the leaf nodes of the HTML tree. Therefore, we define the data type for a leaf node to represent ‘what characteristic of information it carries’. For example, almost all viewable textual content in a webpage is in text nodes. Therefore, a text node can be a child node of different page elements such as a paragraph element `<p>`, a table element `<table>` or a link element `<a>`. Without such characteristics, it would be even difficult for people to find the required information in page variants quickly. The data type describes three kinds of information: property, affiliation and structure.

- **Property** is text, image, video, audio and others.
  1. Text is the character string in webpages such as an article.
  2. Image, video, audio are instances of the image, video, audio multimedia file respectively.
  3. Others are properties that are not text, image, video, or audio.

The property of text, image and others is distinguished by the leaf node name; video, audio is by the leaf node name or the link’s value from the leaf’s ancestor node `<a>`, `<embed>` corresponding to HTML4 and HTML5 standard.

- **Affiliation** is the HTML element type of a leaf node which is determined by ancestor nodes the leaf node affiliated to. It denotes that there exists specific ancestor node(s) of a leaf node, such as `<a>`, `<header>`, `<li>`, `<td>`, `<h1>`, etc., which probably may not be changed in its page variants. For example, if a textual information within a hyperlink (has the ancestor `<a>`) is selected as target, then the corresponding goal is probably still in a link environment in a page variant.

- **Structure** is single occurrence or sequential occurrence.
  1. Single occurrence denotes a leaf node and its ancestor nodes do not have similar sibling nodes.
  2. Sequential occurrence denotes a leaf node and its ancestor nodes have a list of similar sibling nodes.

For example, for the product name of the same product marked in Figure 4, the structure is different in the detail page and the list page. Figure 4(a) is a detail page corresponding to the HTML tree in Figure 1 which focuses on presenting a single object. Figure 4(b) is a list page which contains similar objects in a sequential way.
3.2 Node distance

The node distance depicts the layout relationship between a leaf node and other nodes in an HTML tree. The layout of a webpage is usually displayed in the left-to-right, top-to-bottom order. As illustrated in Figure 4(a) and Figure 1 (a webpage and its HTML tree structure), the path layer of a page element starting from the root node indicates the ‘height’ level of the information carried by the corresponding node in the tree. The highest layers of each path are visible contents of the page, which can be seen as horizontal linear arranged nodes from left-to-right, e.g., leaf nodes from ① to ⑩ in Figure 1. We find the following features between two nodes from horizontal and vertical views.

- The leaf sequence between two nodes of an HTML tree controls the layout distance of the corresponding visible contents of a webpage.
- The path layer of two nodes in the HTML tree controls the inner height distance of the nodes.

Based on these two features, we define the concept of node distance in various levels: leaf level (absolute/relative leaf distances) and path level (path distance). Because the desired information can always be selected by a leaf node, we focus on the distance between a leaf node \( l \) and another node \( n \) in tree \( T \). The idea of measuring the leaf distance between \( l \) and \( n \) is to map \( n \) to a leaf node \( l_n \) (leaf boundary) within a leaf array that linearly consists of all leaf nodes of \( T \). Then we use the difference value between the index of the \( l \) and \( l_n \) in the leaf array to represent the leaf distance between them. The formal definitions of leaf and path distances are given as follows.

Let \( lml(n) \) and \( rmr(n) \) be the leftmost and rightmost leaf nodes of the subtree rooted at node \( n \), which are called boundary nodes of node \( n \). If \( n \) is a leaf node, both boundary nodes are \( n \) itself.

**Definition 3.1:** Let \( m \) be the total number of leaf nodes in \( T \). The leaf array is a linear array \([l_1, l_2, ..., l_m]\) which consists of all leaf nodes of \( T \) ordering by left-to-right, where
Obviously each node $n$ of $T$ can map its boundary nodes into the leaf array. For instance, for the root node $root$, $l_1 = lml(root)$ and $l_m = rml(root)$.

**Definition 3.2:** Let $idx(l)$ and $idx_{relative}(l)$ be indexes of $l$ in the leaf array and relative leaf array respectively. A leaf distance $leaf(l_a, l_b)$ between $l_a$ and $l_b$ is the number of leaf nodes between $l_a$(exclusive) and $l_b$(inclusive). It can be calculated by the following formula.

$$
leaf(l_a, l_b) = idx(l_b) - idx(l_a)
$$

(1)

Correspondingly, a relative leaf distance $leaf_{relative}(l_a, l_b)$ is the number of leaf nodes, which have the same property with $target$, between $l_a$(exclusive) and $l_b$(inclusive).

$$
leaf_{relative}(l_a, l_b) = idx_{relative}(l_b) - idx_{relative}(l_a)
$$

(2)

For a node $l_k$ whose property is different from $target$’s, toward the direction of $Target$, e.g., rightwards, we search the first node $l_{k+i}$ ($k+i < m$) that has the same property as $Target$’s. Then we specify the index of $l_{k+i}$ in the relative leaf array as the relative index of $l_k$, i.e., $idx_{relative}(l_k) = idx_{relative}(l_{k+i})$. For example, a relative leaf array is shown in Figure 5, i.e., $[l_2, l_3, l_5, l_6, l_7]$, that have the same property as $target$’s, i.e., text. The property of $l_1$ is img which is not the same as the text. Along the arrow toward $target$ in the figure, we will get $l_2$ firstly whose property is the same as $target$’s. Therefore, $idx_{relative}(l_1) = idx_{relative}(l_2) = 1$.

**Definition 3.3:** For a leaf node $l$ and a node $n$ in tree $T$, the leaf boundary of $n$, denoted by $LB(l, n)$, is the leaf node in the set of boundary nodes of $n$, i.e., $\{lml(n), rml(n)\}$, which has the smaller absolute value of leaf distance between it and the leaf node $l$.

$$
LB(l, n)=\begin{cases}
lml(n) & |leaf(l, lml(n))| \leq |leaf(l, rml(n))| \\
rml(n) & |leaf(l, lml(n))| > |leaf(l, rml(n))|
\end{cases}
$$

(3)

Based on the above definitions, we define the leaf and path level distances as follows.

**Definition 3.4:** The absolute leaf distance $ALD(l, n)$ in $T$ is the leaf distance between $l$ and $n$’s leaf boundary $LB(l, n)$.

$$
ALD(l, n) = leaf(l, LB(l, n))
$$

(4)

**Definition 3.5:** The relative leaf distance $RLD(l, n)$ in $T$ is the relative distance between $l$ and $n$’s leaf boundary $LB(l, n)$, where leaf nodes between $l$ and $LB(l, n)$ must have the same property of data type with $target$.

$$
RLD(l, n) = leaf_{relative}(l, LB(l, n))
$$

(5)
Definition 3.6: Let $dca(l, n)$ be the deepest common ancestor node, $edge(l, n)$ the number of edges between $l$ and $n$. The path distance $PD(l, n)$ is the number of edges on the path from $l$ to $n$ via the node $dca(l, n)$.

$$PD(l, n) = edge(l, dca(l, n)) + edge(n, dca(l, n))$$

For example, as shown in Figure 5, leaf array $[l_1, l_2, l_3, l_4, l_5, l_6, l_7]$ and relative leaf array $[l_2, l_3, l_5, l_6, l_7]$ are generated. Boundary nodes of $l$ and $n$ are $lml(l) = rml(l) = 6$, i.e., $\{l_6\}$ and $lml(n) = 4$, i.e., $\{l_4, l_5\}$ respectively. Then for $l$, the leaf boundary of $n$, denoted by $LB(l, n)$, is the leaf node $rml(n)$, i.e., $\{l_4\}$, because $|\text{leaf}(l, rml(n))| = |4 - 6| = 2$ which is smaller than $|\text{leaf}(l, lml(n))| = |2 - 6| = 4$. Therefore, we get $ALD(l, n) = \text{leaf}(l, LB(l, n)) = \text{idx}(LB(l, n)) = \text{idx}(l) = \text{idx}(3) - \text{idx}(5) = 4 - 6 = -2$. We get $RLD(l, n) = \text{leaf}_{\text{relative}}(l, LB(l, n)) = \text{leaf}_{\text{relative}}(l_6, l_4) = 3 - 4 = -1$. For $PD(l, n)$, we first get node $dca(l, n) = dca(\{l_6, l_4\})$, i.e., the div pointed by the $dca(l, n)$ in the figure. Then we get $edge(l, dca(l, n)) = 5$ and $edge(n, dca(l, n)) = 1$ by counting the number of edges. Finally, we get $PD(l, n) = edge(l, dca(l, n)) + edge(n, dca(l, n)) = 5 + 1 = 6$.

Figure 5: An example of calculating node distances between a grey-coloured leaf node $l$ and a blue-coloured intermediate node $n$ in a subtree of Figure 1 (see online version for colours)
4 Neighbour zone of designated information

The neighbour zone represents a probable location of the node goal in $T_2$ at the leaf node level, whose definition is presented as follows.

**Definition 4.1:** A neighbour zone is an ordered list of leaf nodes that are symmetrically distributed on both sides of leaf node $C$ within a radius $r$. The leaf node $C$ with index $c$ is called centre node.

\[ \text{NZ}_{(c,r)} = \text{leaf array}[l_{c-r}, \ldots, l_c, \ldots l_{c+r}] \]  

(7)

We use the following pseudo-code with Figure 6 together to depict the general process of neighbour zone acquisition algorithm. We note that the node-distance means a triple of values of absolute leaf distance, relative leaf distance and path distance, i.e.,

\[ \text{node-distance}(l,n) = \langle \text{ALD}(l,n), \text{RLD}(l,n), \text{PD}(l,n) \rangle \], which are shortly denoted as $\langle \text{ALD}, \text{RLD}, \text{PD} \rangle$ if the context is clear. The input are the HTML trees $T_1$, $T_2$ and the target node target in $T_1$. The output is the neighbour zone $\text{NZ}$ that we defined in equation (7).

**Algorithm 1** Neighbour zone of desired information

```
1 begin
2 \hspace{1cm} U \leftarrow D(T_1,T_2);
3 \hspace{1cm} C_k \leftarrow \text{null} (1 \leq k \leq m_2);
4 \hspace{1cm} \text{for } U <i, j> \in U \text{ do}
5 \hspace{2cm} d \leftarrow \text{node-distance}(U <i>, \text{Target});
6 \hspace{2cm} \text{if } \text{node-distance}(U <j>, C_{jk}) == d \text{ then}
7 \hspace{3cm} \text{add } C_{jk} \text{ to } C_k;
8 \hspace{1cm} C \leftarrow \text{select most probable one from } C_k;
9 \hspace{1cm} \text{return } \text{NZ}_{(c,r)} \leftarrow \text{leaf array}[l_{c-r}, \ldots, l_c, \ldots l_{c+r}]
```

1 In line 2, We first compute the unchanged node pair set $U$ by matching tree $T_1$ with $T_2$. Then we initialise a set of centre node candidates $C_k (1 \leq k \leq m_2)$ in line 3, where $m_2$ is the total number of leaf nodes in $T_2$. 

2 Line (4–5) calculate the node-distance $d$ between target and each unchanged node $U <i>$ in $T_1$, i.e., $d$ = node-distance ($U <i>$, target).

3 For each unchanged node $U <j>$ in $T_2$, line (6–7) calculate a set of leaf nodes \{C_{jk}[jk = 1, 2, \ldots m_2]\} having the same distance $d$ from $U <j>$, i.e., node-distance($U <j>$, $C_{jk}$)$=d$.

4 Line 8 selects the most probable one from $C_k (k=1,\ldots,m_2)$ as the centre node $C$. Then, in line 9, let the nodes on two sides of $C$ within the radius $r$ be the neighbour zone $\text{NZ}$. Here, if $c-r < 0$, then let $c-r = 0$; if $c+r > m_2$, then let $c+r = m_2$.

For the illustration shown in Figure 6, suppose that we have an unchanged node pair $U <u_3, u'_3>$. Let node-distance between target $X$ and $u_3$ be $<2, 2, 8>$. Then we look for
the leaf nodes of $T_2$ whose node distance from $u'_3$ is the same. As a result, we have $C_1$ and can consider that the goal node exists in the neighbour zone of $C_1$.

Figure 6  Overview of acquiring the neighbour zone (see online version for colours)

We present the definition and calculation of unchanged node pair in Section 4.1 explaining Step 1; the computation of centre node candidates in Section 4.2 covering Step 2 and Step 3; the selection of the final centre node in Section 4.3 addressing Step 4.

4.1 Unchanged node pairs

We compute the unchanged node pair set using the tree edit distance (TED) algorithm, which is also expressed in terms of tree mapping and widely used to measure the similarity between two trees. The TED is expressed as the minimum cost of transferring tree $T_1$ into $T_2$ through an edit script consists of a sequence of elementary operations: deletion, insertion and change (Zhang and Shasha, 1989). Such edit script will generate a minimum cost tree mapping defined as follows.

**Definition 4.2:** Let $V(T_1)$, $V(T_2)$ be the set of all nodes in $T_1$ and $T_2$, $i_1$, $i_2$ and $j_1$, $j_2$ arbitrary nodes of $T_1$ and $T_2$ respectively, $M$ a set of node pairs from $V(T_1) \times V(T_2)$ which is a tree edit distance mapping between $T_1$ and $T_2$. When the set of node pairs satisfies the following three conditions ($\Leftrightarrow$ denotes if and only if), we call it a mapping.

- $i_1 = i_2 \Leftrightarrow j_1 = j_2$.
- $i_1$ is to the left of $i_2 \Leftrightarrow j_1$ is to the left of $j_2$.
- $i_1$ is an ancestor of $i_2 \Leftrightarrow j_1$ is an ancestor of $j_2$.

As an intuitive explanation, a tree mapping is a set of node pairs which are one-to-one mapped from tree $T_1$ and $T_2$ two sides. It preserves the sibling order, e.g., elements in the left sibling branches of $T_1$ can only be mapped with elements in the left sibling branches of $T_2$. Similarly, the mapping also preserves the ancestor order in the HTML tree, e.g., ‘above’, ‘below’ hierarchy of the path layer.

The mapping can be generated by the tree edit distance algorithm as follows.
Definition 4.3: Let $T_1$, $T_2$ be the rooted ordered labeled trees and $E = o_1, o_2, ..., o_n$ the shortest length edit script that transforms $T_1$ into $T_2$. Let the number of operations of $E$ be $n_d$ (deletion), $n_i$ (insertion) and $n_c$ (change) respectively. Let $\gamma$ be a cost function on operations that $\gamma$(deletion) = $c_1$, $\gamma$(insertion) = $c_2$, $\gamma$(change) = $c_3$. The tree edit distance $D(T_1, T_2)$ is the minimum cost of edit operations needed to transform $T_1$ to $T_2$ calculated as follows.

$$D(T_1, T_2) = c_1 \cdot n_d + c_2 \cdot n_i + c_3 \cdot n_c$$

(8)

The $\gamma$ is set to be a simple unit cost function where $c_1 = c_2 = c_3 = 1$. Therefore, the tree edit distance can be seen as the minimum number of edit operations needed to transform $T_1$ to $T_2$.

We obtain the unchanged node pair set, i.e., matched nodes of two trees, by the following procedures. We use the TED algorithm to compute the edit script. During the TED computation, it stores the left and above nodes for each node, which will generate a mapping between two trees. In other words, the mapping is generated simultaneously with the calculation of the edit script.

Definition 4.4: The unchanged node pair $U$ is a subset of tree mapping $M$ between $T_1$ and $T_2$ satisfying $U = \{<i, j> | T_1[i] = T_2[j]\}$.

More information on the TED algorithm can be found in the original paper (Zhang and Shasha, 1989) which is deemed as the most suitable one for HTML trees because its running time depends on the height of the trees rather than the total number of nodes (Cording and Lyngby, 2011).

4.2 Centre node generation

As the observation mentioned in Section 2, there are some page elements in a page that the relative position (leaf sequence and path layer) between two of them are unchanged in its page variants. Therefore, we use the node-distance (Section 3.2) and unchanged node pair (Section 4.1) to search possible locations of goal on the tree structure. Such locations are represented by the centre node candidate whose definition is given as follows.

Definition 4.5: Let $<u_i, u_j>$ be an unchanged node pair between $T_1$ and $T_2$. For Target $X$ of $T_1$, a centre node candidate is a node $C_k$ ($k=1,2,...$) of $T_2$, which satisfies $\{C_k | C_k$ is a leaf node and node-distance($u_j, C_k$) = node-distance($u_i, X$)\}.

We still use $T_1$ and $T_2$ in Figure 1 as an example. For simplicity, as shown in Table 1, we select nodes $<\text{div}, \text{div}_2>$, $<\text{div}, \text{div}_2>$, $<\text{div}, \text{div}_2>$, $<\text{div}, \text{div}_2>$ and $<\text{div}, \text{div}_2>$ (abbreviation of node $<\text{div id = 'price'}>$) to be the unchanged node pair set, which are illustrated in $U<i>, U<j>$ columns respectively. For Target $\text{div}_2$, we first compute the node-distance, i.e., $<\text{ALD}, \text{RLD}, \text{PD}>$ for each $U<i>$ in $T_1$. Then for the corresponding unchanged node $U<j>$ in $T_2$, we search the nodes in $T_2$ that have the same node-distance which are listed in the fourth column nodes having the same distance. For example, from node $\text{div}_2$, there is only one node, i.e., $\text{div}_2$ that having the same ALD, i.e., $\text{div}_2$. Likewise, the node with the same RLD is $\text{div}_2$. Therefore, for node $\text{div}_2$, there exists none node in $T_2$ having the same node-distance. As a result, we get centre node candidates $C_1 = \text{div}_2, C_2 = \text{div}_2$. 

Stable web scraping

313
4.3 Neighbour zone acquisition

From the above process, we have known that multiple centre node candidates can be generated because:

- each unchanged node pair can generate a unique centre node candidate;
- the same centre node candidate can be generated by multiple unchanged node pairs.

Therefore, the question is how to select the most probable one as the centre node $C$.

Intuitively speaking, we choose the one that is closest to the desired node or is generated the most times.

Let $\text{count}_k$ be the number of how many times of a centre node candidate $C_k$ ($k = 1, 2, \ldots$) generated by unchanged node pairs $<u_1, u_1>, <u_i, u_i>$ ( $i \in [1, \text{count}_k]$), then we select the centre node $C$ by the following two requirements.

- **Near the target.** We first select the one $C$ that has the minimum value of the average $ALD$ between target $X$ and the corresponding $U_{<i>}$ in $T_1$:

$$C = \{C_k | C_k \text{ has minimum } \sum_{i=1}^{\text{count}_k} |ALD(X, u_i)| / \text{count}_k \}.$$  

- **Majority voting.** If still exist multiple $C_k$ that have the same minimum value of the average $ALD$, we select the one $C$ that has the maximum value of the $\text{count}_k$:

$$C = \{C_k | C_k \text{ has maximum } \text{count}_k \}.$$  

For the example shown in Table 1, we get $C_1 = \{5\}$ whose $\text{count}_1 = 1$ and $C_2 = \{6\}$ whose $\text{count}_2 = 2$. The average $ALD$ of $C_1$ and $C_2$ are the same, i.e., $|−1/\text{count}_1 = 1, |0 + 2|/\text{count}_2 = 1$. Then because $C_2$ is larger than $C_1$, we choose the $C_2$, i.e., $\{6\}$ as the centre node $C$.

Table 1  An example of centre node generation

<table>
<thead>
<tr>
<th>$U_{&lt;i&gt;}$</th>
<th>Node-distance</th>
<th>$U_{&lt;j&gt;}$</th>
<th>Nodes having the same distance</th>
<th>$C_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$RLD = −1$</td>
<td></td>
<td>${}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$PD = 9$</td>
<td></td>
<td>${}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\text{inserted, }{5}, {3}, {7}$</td>
<td></td>
<td>${}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$RLD = 0$</td>
<td></td>
<td>${}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$PD = 0$</td>
<td></td>
<td>${}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\text{inserted, }{5}, {3}, {7}$</td>
<td></td>
<td>${}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$RLD = 2$</td>
<td></td>
<td>${}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$PD = 8$</td>
<td></td>
<td>${}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\text{inserted, }{5}, {3}, {7}$</td>
<td></td>
<td>${}$</td>
<td></td>
</tr>
<tr>
<td>div_price</td>
<td>$ALD = −1$</td>
<td>div_price</td>
<td>${}$</td>
<td>${}$</td>
</tr>
<tr>
<td></td>
<td>$RLD = −1$</td>
<td></td>
<td>${}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$PD = 4$</td>
<td></td>
<td>${}$</td>
<td>${}$</td>
</tr>
</tbody>
</table>
After getting the centre node \( C \), we set the radius \( r \) to acquire nodes of the neighbour zone. The \( r \) can be determined by the TED value or the total number of nodes of two trees, while based on the experiment tuning, we set it with an empirical value ten.

5 Path similarity of page elements

Now we focus on searching goal from the neighbour zone. We rank the path similarity between target and nodes of the neighbour zone.

Firstly, we filter out nodes whose property is different from target’s. Then we compute the scores of the remaining nodes, which are measured from three aspects: tag path, attribute path, affiliation. In addition, we use the list order if the structure of the desired information is sequential. Finally, we normalise these scores and select the node having the highest score as goal. If more than one candidate corresponds to the highest score, we select them together.

We define the following format to describe an XPath information: 

\[
N_1[O_1]/\cdots/N_n[O_n]/\cdots
\]

where \( N_n \) is the HTML node name of the \( n \)th node; \( [O_n] \) is the order of the \( n \)th node among its siblings having \( N_n \) as names; \( A_{n-m} \) is the name of the \( m \)th attribute of the \( n \)th node; \( V_{n-m} \) is the corresponding value of attribute \( A_{n-m} \); and \( N_n \) is the parent node of \( N_{n+1} \). Because \( O_n \) and \( A_{n-m} \) are neglected if there are no siblings or attributes or they are not fixed in the node searching, we divide this XPath into the tag path and the attribute path for further processing. For emphasizing the differentiation between paths, we omit the head part which every path contains, i.e., html and body tags and their attributes.

5.1 Tag path similarity

Following our XPath definition, the tag path is defined as 

\[
N_1[O_1]/\cdots/N_n[O_n]/\cdots
\]

which uniquely describes the position of a node in an HTML tree. For instance, the tag path of leaf node ⑥ in Figure 1 can be denoted as (omit html, body and the same below):

\[
P_a = /\text{div}[2]/\text{div}[2]/\text{table}[1]/\text{td}[2]/\text{text}()
\]

We observe that page variants often were caused only by the sibling order of the desired information had been changed. Therefore, if two paths have the same sequence of tags (ignoring the difference of orders among siblings), we can use this feature to distinguish them with others that do not have the same tag sequence. We define such tag path without sibling order, namely tag path cut (cut the order number), as \( N_1/\cdots/N_n/\cdots \) to distinguish the tag sequence. For example, as shown in Figure 1, leaf nodes ⑤~⑦ and ③~⑬ share the same tag path cut \( P_{\text{cut}} \) shown below.

\[
P_{a_{\text{cut}}} = /\text{div}/\text{div}/\text{table}/\text{td}/\text{text}()
\]

5.1.1 Tag path edit distance score (TPED)

The string edit distance (SED) is the way of quantifying how dissimilar two strings are to one another by counting the minimum number of basic operations (deletion, insertion,
change) required to transform one string into the other. Since it can be seen as a special case of the TED algorithm which was introduced in Section 4.1, here we directly refer to the function of the SED algorithm on account of paper space, while the detailed algorithm can be found in paper (Wagner and Fischer, 1974). Let \( \delta(s_a, s_b) \) be the string edit distance of string \( s_a \) and \( s_b \), which can be seen as the minimum number of edit operations needed to transform \( s_a \) to \( s_b \). We treat each tag name in the tag path as a character in the string, we can get the tag path edit distance score \( TPED(P_a, P_b) \) between path \( P_a \) and \( P_b \):

\[
TPED(P_a, P_b) = 1 - \frac{\delta(P_a, P_b)}{\max(|P_a|, |P_b|)}
\]

where \(|P_a|, |P_b|\) denote the number of tags in \( P_a \), \( P_b \) respectively. \( TPED \) score reaches its best value at 1 (identical paths) and worst at 0.

For example, in Figure 1, let the XPath of node (6) be \( P_a \) and node (5) be \( P_b \) whose tag path and tag path cut are shown as follows.

- \( P_a = /div[2]/div[2]/table/tr[1]/td[2]/text() \)
- \( P_b = /div[2]/div[2]/table/tr[1]/td[1]/text() \)
- \( P_{a, cut} = /div/div/table/tr/td[text()] \)
- \( P_{b, cut} = /div/div/table/tr/td[text()] \)

The tag path edit distance \( \delta(P_a, P_b) = 1 \) (replace \( td[2] \) to \( td[1] \)) and \( \delta(P_{a, cut}, P_{b, cut}) = 0 \). All lengths of paths are 6. Therefore, we get \( TPED(P_a, P_b) = 1 - \frac{1}{6} = 0.833 \), \( TPED(P_{a, cut}, P_{b, cut}) = 1 - \frac{0}{6} = 1 \).

### 5.1.2 Common tag path distance score

Intuitively, the common tag path distance (CTPD) calculates the ratio of how much the continuous common part between two paths from the beginning of paths. As illustrated in Figure 7, the common path \( CP(P_a, P_b) \) between two tag paths \( P_a \), \( P_b \) are the identical tags that start from the head of paths continuously. The \( Diff_a \), \( Diff_b \) are the remaining subpath that \( P_a \), \( P_b \) remove \( CP(P_a, P_b) \) respectively. We denote them as follows.

- \( P_a = CP(P_a, P_b) \cup Diff_a \)
- \( P_b = CP(P_a, P_b) \cup Diff_b \)

The CTPD is defined as follows, which reaches its best value at 1 and worst at 0 (no common tag path).

\[
CTPD(P_a, P_b) = \frac{|CP(P_a, P_b)|}{|CP(P_a, P_b)| + |Diff_a| + |Diff_b|}
\]

We still use node (6) (\( P_a \)) and node (5) (\( P_b \)) in Figure 1 as an example, whose tag path and tag path cut are listed in Section 5.1.1. Then \( CP(P_a, P_b) = /div[2]/div[2]/table/tr[1] \), the corresponding \( Diff_a = /td[2]/text() \), \( Diff_b = /td[1]/text() \). The \( CP(P_{a, cut}, P_{b, cut}) = /div/div/table/tr/td[text()] \) whose
corresponding $\text{Diff}_a$ and $\text{Diff}_b$ are empty (length $= 0$). Therefore, $\text{CTPD}(P_a, P_b) = \frac{4}{4+2+2} = 0.5$, $\text{CTPD}(P_a, \text{cut}, P_b, \text{cut}) = 1$.

**Figure 7** Common tag path distance $\text{CP}(P_a, P_b)$ (see online version for colours)

In summary, we compute the final tag path similarity score, denoted by $\text{TAG}(P_a, P_b)$, as the following formula. For brevity, let $\text{TPED}$ represent $\text{TPED}(P_a, P_b)$, $\text{TPED}_{\text{cut}}$ represent $\text{TPED}(P_a, \text{cut}, P_b, \text{cut})$, as well as $\text{CTPD}$ and $\text{CTPD}_{\text{cut}}$. Let $e_1, e_2, e_3, e_4$ be weight coefficients of these four tag path scores respectively.

- If the structure is single

\[
\text{TAG}(P_a, P_b) = \frac{e_1 \cdot \text{TPED} + e_2 \cdot \text{TPED}_{\text{cut}} + e_3 \cdot \text{CTPD} + e_4 \cdot \text{CTPD}_{\text{cut}}}{\sum_{i=1}^{4} e_i}
\]  

- If the structure is sequential

\[
\text{TAG}(P_a, P_b) = \frac{e_2 \cdot \text{TPED}_{\text{cut}} + e_4 \cdot \text{CTPD}_{\text{cut}}}{\sum_{i=1}^{2} e_i}
\]

For the sake of generality, we set these weights to be 1. It is possible to tune special parameters for different websites. Then following the example of $\text{6} (P_a)$ and $\text{5} (P_b)$ in Figure 1 (the structure is single), the $\text{TAG}(P_a, P_b) = \frac{0.833+1+0.5+1}{4} = 0.833$.

### 5.2 Attribute path similarity

Each element of an XPath can have attributes which are used to define the characteristics of an HTML page element. An attribute is added into the start tag of an element, which usually appears as a name-value pair separated
In attribute of one layer by a number. Because there are multiple attribute values within one path layer, we specify the unit class="product" set the unit attribute score = 3 of a node, then attributes of the W number of its attribute layers, Let attributes of the base attribute path can be matched with that of other attribute paths.

The similarity between two attribute paths is based on the ratio of matching the attributes of a base attribute path with another’s. For example, the attribute path of node ⑥ in Figure 1 is as follows, where the ‘_‘ denotes the attribute is empty.

- \( AP_a = /id="container"/id="price"/id="main" \text{class="product"/.../class="price"/text="¥8080"} \)

Then the path layer of the first attribute element (id="container") is 1 and the layer of the last one (text = “¥8080”) is 6. We observe that the attribute of lower layer contains very general information and shared by many descendants, while the higher the path layer is, the more distinctive and exclusive the attribute is. Therefore, we provide larger weight factors for the attributes of higher path layers, i.e., closer to leaf nodes.

Accordingly, we develop the function \( ATTR(AP_a, AP_b) \) in equation (13) to evaluate similarity between two attribute paths \( AP_a \) and \( AP_b \), whose output is a score which gets its best value at 1 and worst at 0. The function focuses on measuring how many unit attributes of the base attribute path can be matched with that of other attribute paths. Let \( AP_b \) be the base attribute path to be compared with. Then for \( AP_a \), let \( n \) be the number of its attribute layers, \( m_i \) the number of unit attribute of the \( i \)th layer of \( AP_a \), \( W_i \) the weight of the \( i \)th layer of \( AP_a \). If we specify a number \( i \) in the attribute path of a node, then attributes of the \( i \)th layer are determined. For example, if specifying \( i = 3 \) in \( AP_a \) of Figure 8, we will get two attributes of the 3rd layer, i.e., \( id="main" \) and \( \text{class="product"} \). We call such attribute of a single name-value pair as an unit attribute. Because there are multiple attribute values within one path layer, we specify the unit attribute of one layer by a number \( j \). Let \( attr(AP_i, i, j) \) be the \( j \)th unit attribute of path layer \( i \) of the attribute path \( AP \). For example, \( attr(AP_a, 3, 2) = \text{class="product"} \). The unit attribute in the same path layer will share the weight of the layer. Based on our observation, we set the weight of each path layer to be its layer index, i.e., \( W_i = i \). For example, both unit attributes \( id="main" \) and \( \text{class="product"} \) in layer 3 (\( W_3 = 3 \)) are \( \frac{3}{2} \).

If the \( attr(AP_a,i,j) \) (base) finds anyone matching within all unit attributes of \( AP_b \), we set the unit attribute score \( unitAttrScore(AP_a, i, j, AP_b) \) be 1, otherwise, be 0.

\[
ATTR(AP_a, AP_b) = \frac{\sum_{i=1}^{n} \sum_{j=1}^{m_i} \frac{W_i}{m_i} \cdot unitAttrScore(AP_a, i, j, AP_b)}{\sum_{i=1}^{n} W_i} \quad (13)
\]

In real-world webpages, judging whether two unit attributes are matched is not a trivial task. For example, unit attributes \( id="bestPriceSept" \) and \( id = "best\_price\_09" \) do not have the same string representation but may have the same meaning. We calculate the \( unitAttrScore \) between two unit attributes in the following way to make the approximate match.

- If attribute names of two unit attributes are different, a similarity score is set to 0.
- If attribute name is id or class, first we segment the attribute values into words and then compute the \( Jaccard \) coefficient (intersection over union) as the similarity score. The thesaurus is used here for judging whether two single words are matched.
If attribute name is `href` or `src`, we clean and filter the URL string first and then compute the SED value as the similarity score.

For other attribute names, we compute the SED value as the similarity score.

At last, if the similarity score passes a pre-defined threshold, we treat two unit attributes as a match and set the unit attribute score, i.e., `unitAttrScore`, to be 1, otherwise 0.

Depending on the structure of the data type of the target node, i.e., single or sequential, there are some differences in the parse, clean and match policy processes. For example, if the structure is sequential, the last anchor value in the attribute path will be deleted; only the first appeared domain value is picked as for the `href` and `src` attribute; different thresholds are chosen for judging whether two unit attributes are similarity and etc.

Figure 8 shows an example of calculating attribute path similarity. We use the attribute paths of node \( \overline{6} \) (base \( AP_a \)) and node \( \overline{5} \) (\( AP_b \)) as an example. The base attribute path \( AP_a \) has six layers, i.e., \( n = 6 \). The weight of each layer is equal with its layer value. Therefore, the sum of the weights of path layers is \( \sum_{i=1}^{6} W_i = 21 \).

Obviously, except the 5th and 6th of \( AP_a \) whose attribute values are written in bold, all unit attributes of \( AP_a \) can get a matched one in \( AP_b \). Therefore, the final attribute similarity score of the two paths is:

\[
\frac{1 \times 1 + 2 \times 1 + 3 \times 2 + 4 \times 1 + 0 + 0}{21} = \frac{10}{21} = 0.476.
\]

Figure 8  An example of the attribute path with path layers (see online version for colours)

5.3 List order similarity

When the target node is located in the sequential structure, e.g., the product name of the first item in Figure 4(b), sometimes it will be difficult to distinguish it with other siblings through the similarity of tag and attribute path because they are almost the same with each other. Therefore, we need to identify the desired node by the order it is positioned in the sequential structure. An example is illustrated in Figure 9 where the \( li[i] \) \((i = 1, 2, ..., 9)\) denotes the \( i \)-th anchor of HTML list structure \( li \); the similar siblings (big grey nodes) have almost the same tag path and attribute scores with the designated node (big black node). Let the node labelled designated node be target which is in the subtree rooted at the third branch of the list, i.e., \( li[3] \), then goal in a page variant should also be near the third branch in the corresponding sequential structure.

We use the list order score to measure the similarity of the order of two nodes in the sequential structure by their tag paths. Suppose we have a node with tag path \( P_a \) in \( T_1 \) and another node with \( P_b \) in \( T_2 \). Intuitively, for comparing the position of two nodes in their sequential structures, the key is to respectively find the node (namely fertile node) in their tag paths that started the sequential structure, and then to acquire the list order of each node within the structure. We define the list order of a node as \( \frac{SIB}{NUM} \) where the \( SIB \) is the ordering number of the node in its siblings and the \( NUM \) is the number of
siblings in the list structure. In Figure 9, the fertile node starts a sequential structure which has 9 siblings \((NUM = 9)\) and the designated node is on the third sibling branch \((SIB = 3)\), then we get the list order of the designated node is \(\frac{3}{9}\).

**Figure 9** List order of a designated node in the sequential structure (see online version for colours)

![Diagram of list order](image)

After obtaining the list order of two nodes, we compute the list order similarity score \(ORDER(P_a, P_b)\) by equation (14) which reaches its best value at 1 and worst at 0. Let the list order of two nodes be \(\frac{SIB_a}{NUM_a}, \frac{SIB_b}{NUM_b}\) respectively, \(\max(NUM_a, NUM_b)\) the largest value of \(NUM_a\) and \(NUM_b\).

\[
ORDER(P_a, P_b) = \left(1 - \frac{|SIB_a - SIB_b|}{\max(NUM_a, NUM_b)}\right) \cdot \left(1 - \frac{|NUM_a - NUM_b|}{\max(NUM_a, NUM_b)}\right)
\] (14)

For example, the tag path of the designated node in Figure 9 be \(P_a\), then the list order is \(\frac{SIB_a}{NUM_a} = \frac{3}{9}\). Suppose we have a page variant where only one new branch was inserted. We select the 3rd sibling of page variant as \(P_b\), whose list order is \(\frac{3}{10}\). Then the list order similarity score \(ORDER(P_a, P_b)\) is \((1 - \frac{|3 - 3|}{\max(9, 10)}) \cdot (1 - \frac{|9 - 10|}{\max(9, 10)}) = 1 \cdot 0.9 = 0.9\).

5.4 **Affiliation similarity and final score**

At last, we compute the affiliation score, denoted by \(AFFI(P_a, P_b)\), simply by judging whether tag paths \(P_a, P_b\) of two nodes contain the same affiliation value. If it contains, the score is 1, otherwise the score is 0.

Let \(XP_a, XP_b\) be the XPath of node \(a\) and \(b\). The path similarity score function \(PATHS(XP_a, XP_b)\) reaches its best value at 1 and worst at 0. We obtain the tag path \(P_a, P_b\) and attribute path \(AP_a, AP_b\) from \(XP_a, XP_b\) respectively. For brevity, let \(TAG\) be \(TAG(P_a, P_b)\), \(ATTR\) be \(ATTR(AP_a, AP_b)\), \(AFFI\) be \(AFFI(P_a, P_b)\), \(ORDER\) be
Stable web scraping

ORDER($P_a$, $P_b$). Let the $Avg$ be a mean function, the final path similarity score is computed with the following formula.

$$PATHS(XP_a, XP_b) = Avg(TAG + ATTR + AFFI + ORDER)$$

(15)

We select the node(s) with the highest path similarity score as the Goal node(s).

6 Experiment and evaluation

In this section, we set up experiments on real-world web data extraction tasks to verify the following research questions in Section 6.1. The XPath-based extraction (Robie et al., 2014) is the baseline method.

6.1 Research questions

RQ1 Does the neighbour zone and path similarity-based method have higher stability than the XPath method on extracting the desired information in page variants?

RQ1-1 With the centre node of the neighbour zone component alone, can we get a more stable extraction result than the XPath method?

RQ1-2 With the path similarity component alone, can we scrape the desired data more stably than the XPath method?

RQ1-3 When combining neighbour zone and path similarity components, can we get a more stable extraction result than using two components alone?

RQ2 If the page variant contains larger structural changes, can the neighbour zone and path similarity-based method extract the desired information stably?

In order to measure the stability mentioned in RQ1 and RQ2, we adopt the most commonly used evaluation criteria in this area, the F1-score (a weighted average of the precision and recall), as performance metrics. An F1-score reaches its best value at 1 and worst at 0. If a method has a higher F1-score on the dataset, it is regarded as more stable.

6.2 Datasets and experiment process

We prepared two datasets. The first one, denoted by Dataset1, is gathered from a large dataset with ground-truth, publicly available at http://swde.codeplex.com/, published by Hao et al. (2011). Because the source files of the dataset are very large (about to 8 GB), we use random sampling technique to facilitate the assembly of the sample. As summarised in Table 2, we randomly selected 101 pages (1 webpage and 100 page variants) for each website to get totally 8,080 webpages from 80 diverse websites in eight verticals such as autos, books, etc.
Table 2 The composition of the Dataset1

<table>
<thead>
<tr>
<th>Vertical</th>
<th>#Sites</th>
<th>#Pages</th>
<th>Target data</th>
<th>Websites</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autos</td>
<td>10</td>
<td>1,010</td>
<td>model, price, engine, fuel-economy</td>
<td>aol, autobytel, automotive, autoweb, carquotes, cars, kbb, motortrend, msn, yahoo, abebooks, barnesandnoble, bookdepository, booksamillion, borders, christianbook, deepdiscount, waterstones</td>
</tr>
<tr>
<td>Books</td>
<td>10</td>
<td>1,010</td>
<td>title, author, ISBN-13, publisher, publish-date</td>
<td>amazon, beachaudio, buy, compsource, ecost, jr, newegg, onsale, penation, theenerds, careercircle, dice, hotjobs, job, jobcircle, jobtarget, monster, nettemps, rightjobs, techcentric, allmovie, amctv, boxofficemojo, hollywood, iheartmovies, imdb, metacritic, rottentomatoes</td>
</tr>
<tr>
<td>Cameras</td>
<td>10</td>
<td>1,010</td>
<td>model, price, manufacturer</td>
<td></td>
</tr>
<tr>
<td>Jobs</td>
<td>10</td>
<td>1,010</td>
<td>title, company, location, date</td>
<td>careerbuilder, dice, hotjobs, job, jobcircle, jobtarget, monster, nettemps, rightjobs, techcentric, allmovie, amctv, boxofficemojo, hollywood, iheartmovies, imdb, metacritic, rottentomatoes</td>
</tr>
<tr>
<td>Movies</td>
<td>10</td>
<td>1,010</td>
<td>title, director, genre, rating</td>
<td>espn, fanhouse, foxsports, msnca, nba, si, slam, usatoday, wiki</td>
</tr>
<tr>
<td>NBA players</td>
<td>10</td>
<td>1,010</td>
<td>name, team, height, weigh</td>
<td>foders, frommers, gayot, opentable, pickarestaurant, restaurantica, tripadvisor, urbanspoon, usdiners, zagat</td>
</tr>
<tr>
<td>Restaurants</td>
<td>10</td>
<td>1,010</td>
<td>name, address, phone, cuisine</td>
<td>collegeboard, collegenavigator, collegeprowler, collegeprowler, ecampusdours, embark, matchcollege, princeontreview, studentaid, usnews</td>
</tr>
<tr>
<td>Universities</td>
<td>10</td>
<td>1,010</td>
<td>name, phone, website, type</td>
<td></td>
</tr>
</tbody>
</table>

Note: The information of domains and target data of each vertical is shown along with the number of samples.

As shown in Table 3, the second dataset, denoted by Dataset2, we manually recorded the failed samples from a large number of extraction experiments. The pages were gathered from internet archive (https://archive.org/) which constantly store historical snapshots of the webpage. Dataset2 contains 90 webpages (30 webpages and 60 page variants with a time interval of 1~77 months) from 22 websites covering eight verticals such as news, travel agent and so on. Because the internet archive cannot collect everything from every website, when the volunteer collects webpage and its page variants, only if the webpage carrying the target information exists in the archive will be selected and saved. Therefore, the integrity of the page in Dataset2 is guaranteed. Please note that the Dataset2 only consists of pages from which the initial XPath failed to extract the target data in page variants. Therefore, the stability test on this dataset will be more challenging. Dataset2 is also used to answer RQ2, because we consider that page variants produced after a long time interval, e.g., several or tens of months after, include larger changes.
<table>
<thead>
<tr>
<th>Website</th>
<th>Initial page 1</th>
<th>Page variant 1–2</th>
<th>Page variant 2–3</th>
<th>Page variant 1–3</th>
<th>Variant type</th>
<th>Page type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN economy</td>
<td>20140101</td>
<td>15 m 0.427  ×  ✓  ✓</td>
<td>18 m 0.461  ✓  ✓  ✓</td>
<td>32 m 0.494  ✓  ✓  ✓</td>
<td>Temporal</td>
<td>Detail</td>
</tr>
<tr>
<td>BBC business</td>
<td>20150101</td>
<td>3 m 0.449  ×  ✓  ✓</td>
<td>17 m 0.074  ✓  ✓  ✓</td>
<td>20 m 0.461  ×  ✓  ✓</td>
<td>Temporal</td>
<td>Detail</td>
</tr>
<tr>
<td>Yahoo! News</td>
<td>20141020</td>
<td>18 m 0.504  ✓  ✓  ✓</td>
<td>3 m 0.331  ✓  ✓  ✓</td>
<td>21 m 0.516  ✓  ✓  ✓</td>
<td>Temporal</td>
<td>Detail</td>
</tr>
<tr>
<td>Booking.com</td>
<td>20151120</td>
<td>4 m 0.146  ✓  ✓  ✓</td>
<td>4 m 0.209  ✓  ✓  ✓</td>
<td>8 m 0.289  ✓  ✓  ✓</td>
<td>Mix</td>
<td>Detail</td>
</tr>
<tr>
<td>Ctrip Cruise</td>
<td>20150721</td>
<td>10 m 0.236  ×  ×  ×</td>
<td>2 m 0.075  ✓  ✓  ✓</td>
<td>12 m 0.231  ×  ×  ×</td>
<td>Mix</td>
<td>List</td>
</tr>
<tr>
<td>Lvmama Cruise</td>
<td>20151118</td>
<td>6 m 0.319  ✓  ✓  ✓</td>
<td>3 m 0.384  ×  ×  ×</td>
<td>7 m 0.377  ✓  ✓  ✓</td>
<td>Mix</td>
<td>Detail</td>
</tr>
<tr>
<td>Lvmama Tour</td>
<td>20160603</td>
<td>2 m 0.435  ×  ×  ×</td>
<td>10 m 0.307  ✓  ✓  ✓</td>
<td>12 m 0.459  ×  ×  ×</td>
<td>Mix</td>
<td>Detail</td>
</tr>
<tr>
<td>Ly Tour</td>
<td>20160229</td>
<td>2 m 0.070  ×  ✓  ✓</td>
<td>10 m 0.276  ×  ×  ×</td>
<td>12 m 0.263  ×  ×  ×</td>
<td>Mix</td>
<td>Detail</td>
</tr>
<tr>
<td>Ly Drive</td>
<td>20160602</td>
<td>2 m 0.174  ×  ✓  ✓</td>
<td>6 m 0.260  ✓  ✓  ✓</td>
<td>8 m 0.329  ✓  ✓  ✓</td>
<td>Mix</td>
<td>Detail</td>
</tr>
<tr>
<td>Ly Visa</td>
<td>20160225</td>
<td>1 m 0.387  ×  ×  ×</td>
<td>3 m 0.366  ✓  ✓  ✓</td>
<td>4 m 0.355  ×  ×  ×</td>
<td>Mix</td>
<td>Detail</td>
</tr>
<tr>
<td>Lvmama Cruise</td>
<td>20160625</td>
<td>1 m 0.387  ×  ×  ×</td>
<td>3 m 0.366  ✓  ✓  ✓</td>
<td>4 m 0.333  ×  ×  ×</td>
<td>Mix</td>
<td>Detail</td>
</tr>
<tr>
<td>Worldbank.org</td>
<td>20140102</td>
<td>5 m 0.066  ✓  ✓  ✓</td>
<td>18 m 0.408  ✓  ✓  ✓</td>
<td>23 m 0.415  ✓  ✓  ✓</td>
<td>Temporal</td>
<td>Detail</td>
</tr>
<tr>
<td>Wikipedia.org</td>
<td>20140104</td>
<td>9 m 0.095  ✓  ✓  ✓</td>
<td>17 m 0.187  ✓  ✓  ✓</td>
<td>26 m 0.244  ✓  ✓  ✓</td>
<td>Temporal</td>
<td>Detail</td>
</tr>
<tr>
<td>BBC Country</td>
<td>20040402</td>
<td>20 m 0.156  ✓  ✓  ✓</td>
<td>57 m 0.310  ✓  ✓  ✓</td>
<td>77 m 0.332  ✓  ✓  ✓</td>
<td>Temporal</td>
<td>Detail</td>
</tr>
<tr>
<td>Github.com</td>
<td>20140517</td>
<td>9 m 0.048  ✓  ✓  ✓</td>
<td>18 m 0.155  ✓  ✓  ✓</td>
<td>27 m 0.174  ✓  ✓  ✓</td>
<td>Temporal</td>
<td>List</td>
</tr>
<tr>
<td>Zhihu.com</td>
<td>20150317</td>
<td>9 m 0.048  ✓  ✓  ✓</td>
<td>18 m 0.155  ✓  ✓  ✓</td>
<td>27 m 0.174  ✓  ✓  ✓</td>
<td>Temporal</td>
<td>List</td>
</tr>
<tr>
<td>Amazon.co.jp</td>
<td>20160108</td>
<td>7 m 0.211  ×  ✓  ✓</td>
<td>10 m 0.211  ✓  ✓  ✓</td>
<td>17 m 0.238  ✓  ✓  ✓</td>
<td>Temporal</td>
<td>Detail</td>
</tr>
<tr>
<td>Ebdar.com</td>
<td>20160108</td>
<td>7 m 0.211  ✓  ✓  ✓</td>
<td>10 m 0.211  ✓  ✓  ✓</td>
<td>17 m 0.238  ✓  ✓  ✓</td>
<td>Temporal</td>
<td>Detail</td>
</tr>
<tr>
<td>Youtube.com</td>
<td>20140617</td>
<td>6 m 0.302  ✓  ✓  ✓</td>
<td>21 m 0.301  ✓  ✓  ✓</td>
<td>27 m 0.378  ✓  ✓  ✓</td>
<td>Mix</td>
<td>Detail</td>
</tr>
<tr>
<td>Eonine.com</td>
<td>20150108</td>
<td>21 m 0.283  ✓  ✓  ✓</td>
<td>16 m 0.179  ✓  ✓  ✓</td>
<td>37 m 0.324  ✓  ✓  ✓</td>
<td>Mix</td>
<td>Detail</td>
</tr>
<tr>
<td>Csdn.net</td>
<td>20150102</td>
<td>15 m 0.209  ✓  ✓  ✓</td>
<td>4 m 0.238  ✓  ✓  ✓</td>
<td>19 m 0.250  ✓  ✓  ✓</td>
<td>Mix</td>
<td>Detail</td>
</tr>
<tr>
<td>Blog.sina.com</td>
<td>20140602</td>
<td>16 m 0.117  ✓  ×  ✓</td>
<td>26 m 0.189  ✓  ✓  ✓</td>
<td>42 m 0.225  ✓  ✓  ✓</td>
<td>Temporal</td>
<td>Detail</td>
</tr>
<tr>
<td>Twitter.com</td>
<td>20150318</td>
<td>11 m 0.200  ✓  ✓  ✓</td>
<td>8 m 0.051  ✓  ✓  ✓</td>
<td>19 m 0.198  ✓  ✓  ✓</td>
<td>Temporal</td>
<td>List</td>
</tr>
</tbody>
</table>

Note: It only consists of pages that the initial XPath failed to repeatedly extract the target data in page variants.
The experiment process is shown in Figure 10. We first construct correct sets of goal nodes (ground truth) for two datasets. The ground truth of Dataset1 was published by Hao et al. (2011). The correct goal nodes of Dataset2 were screened by our volunteers, including two professional crawler developers and four college students having no web scraping experience. For each page of Dataset1, there are 3~5 target nodes specified, and for each target node of a webpage, we can get a goal node in its variant page. Totally, we have extracted 31,400 target nodes from the Dataset1. For Dataset2, we extracted 90 target nodes from Dataset2 totally. There are 1~3 target nodes specified for each page, which the XPath failed to extract.

To compare the stability (precision, recall and F1-score) of main components in our approach with the baseline XPath-based extraction, we implemented three versions of the proposed system as follows.

- **SSN** (stable scraping based on neighbour zone) is the version which directly uses the centre node of the neighbour zone as the goal node. It is only the output of the Section 4.
- **SSP** (stable scraping based on path similarity) is the version which only uses top ranking node(s) of all leaf nodes as the goal node. It is only the output of the Section 5.
- **SSNP** (stable scraping based on neighbour zone and path similarity) is the version which searches the goal node in nodes around the centre node of neighbour zone using the path similarity. It actually is the output of SSN plus the result of SSP.
We implemented a prototype of the method in Java (JDK 1.7). All experiments are run on an Intel Core i5 2.4 GHz PC running Windows 10 64 bit with 8 GB RAM. We use the Jsoup (ver.1.7.2) (https://jsoup.org/) to parse each HTML page into an ordered labeled tree. The XPath of the target node is generated by the widely used tool firepath (https://addons.mozilla.org/en-us/firefox/addon/firepath/), which is an absolute or a relative path.

6.3 Performance metrics

Let a dataset be a finite set of webpages \( W = \{w_1, \ldots, w_i, \ldots, w_n\} \); \( X(w_{ij}) \) the correct goal node in the page \( w_i \) to the \( j \)th target node; \( Y(w_{ij}) \) the output node(s) that were extracted by the method corresponding to target node; \( k_i \) the number of target nodes we given in the page \( w_i \) \((i = 1, \ldots, n)\). For Dataset1, \( n = 8,000 \) and \( 3 \leq k_i \leq 5 \) \((i = 1, 2, \ldots, 8,000)\). For Dataset2, \( n = 60 \) and \( 1 \leq k_i \leq 3 \) \((i = 1, 2, \ldots, 60)\). Then the precision, recall and the F1-score are calculated by the formula as follows.

\[
\text{Precision} = \frac{\sum_{i=1}^{n} \sum_{j=1}^{k_i} |X(w_{ij}) \cap Y(w_{ij})|}{\sum_{i=1}^{n} \sum_{j=1}^{k_i} |Y(w_{ij})|}
\]

\[
\text{Recall} = \frac{\sum_{i=1}^{n} \sum_{j=1}^{k_i} |X(w_{ij}) \cap Y(w_{ij})|}{\sum_{i=1}^{n} \sum_{j=1}^{k_i} |X(w_{ij})|}
\]

\[
\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}
\]

6.4 Experiment results

In Figures 11 and 12, we summarise the precision, recall and F1-score of XPath, SSN, SSP and SSNP in Dataset1 and Dataset2 respectively.

Figure 11 The precision, recall and F1 score of Dataset1 (see online version for colours)
Table 3 also records all extraction result of Dataset2. Each test was started with the ‘Initial page 1’ where users selected the desired data named as ‘target data’. The ‘Page variant #a–#b’ is a page variant ‘#b’ which was changed from the page ‘#a’ and the ‘interval’ means the time interval (month) from ‘#a’ to ‘#b’. The TED value is to quantify how similar two pages are to one another. It reaches its best value at 0 (same page) and worst at 1. The ‘variant type’ is the reason for the variation of the page which includes temporal change and the mix of the template and the temporal change. For example, the future changed version of the template variant of the original page. The ‘page type’ means the detail page or list page. The results of extraction based on each method are recorded as correct and incorrect which are marked as X and × respectively. When multiple results are generated, an underline will be added to the corresponding result symbol. We notice that all XPaths worked on the page ‘#a’ but failed to extract the target data in the page variant ‘#b’.

We have the following observations drawn from these experiments.

- Even a slight structural change can lead to an absolute XPath failure, and when the page structure changed largely the carefully evaluated relative XPath went to fail.
- For most sites, the internal structure of the page changes more as time goes on, which makes it more difficult for the XPath.
- The temporal and mixed type of page variants usually is more challenging for extraction.
- Experiment results of verticals on Dataset1 are shown in Figure 13. We found that few samples consisted of page variants caused by inconsistent templates in Dataset1, where SSN, SSP have a better result than SSNP.
Stable web scraping

1 The reason of the former is that some pages use a large list structure to display information such as current price, discount price, historical prices that have the same path score which leads SSNP to generated multiple results. Although this has been mitigated by using list order similarity (Section 5.3) to distinguish order sequences, for identifying the target data, we still need a more semantic level comparison.

2 The latter is because some webpages swap two large list structures. Because of the inherent limitations of the TED algorithm, the SSN method yields erroneous results, which affects the SSNP results.

- When page variants change largely, e.g., temporal change or the mixed type of change in Dataset2, the SSN and SSP components are not as effective as they performed in the template changed page variants, which indicate the combination approach of SSNP is necessary.

Figure 13 The precision, recall and F1 score of verticals of Dataset1 (see online version for colours)

6.5 Discussion

6.5.1 Answer to the research questions

From Figure 11, we see that in the Dataset1 experiment, The SSN, SSP and SSNP increase the stability score of XPath by 3.3%, 12.7%, 14% respectively. From Figure 12, because the Dataset2 contains only page variants that the XPath method failed to extract, the F1 score of XPath is 0, while SSN is 0.644, SSP is 0.636 and SSNP is 0.891 (the base value is 0, the growth rate is not calculated). From this experimental result, the answers to the question RQ1 (RQ1-1, RQ1-2, RQ1-3) are yes.

From Figure 12 and Table 3, we can see that as the time span increases, the page change grows largely. In this case, the XPath becomes fragile, while SSN reaches the
passing level (0.644) and SSNP still maintains the F1-score above 0.89. Therefore, for the question RQ2, our answer is Yes.

From Figure 13, we also found that our method has a higher stability than (Hao et al., 2011) in results of all the verticals (0.961 vs. 0.844 average). We note the difference of application scenarios is that our method needs to specify a target node for each site without the help of visual files, while Hao et al. (2011) only needs to specify a target node for sites of a vertical but requires visual information support.

6.5.2 External validity

We discuss the threats to external validity of the experimental conclusions. The following aspects of our sample pages in datasets show the generalization of our experiment, which means it would hold for other persons in other places and at other times.

- The samples are from 102 websites having high hit counts covering 16 verticals, such as amazon.com of camera, booking.com of travel, etc.
- More than 30,000 target nodes have been extracted where multiple target nodes were selected from different positions in a page, e.g., product title, price, manufacturer, etc.
- Page variants of long time interval (1 month~77 months) and page variants of main type of data rich pages (detail page and list page) were collected by volunteers.

Therefore, we can consider the datasets we used are sufficiently generalised to keep external validity.

6.5.3 Input requirements

All inputs our approach needs are simply consisted of a HTML source file with designated target nodes, a property switcher and optionally a structure switcher. We note that such inputs are only required on the first use. The property switcher is a Boolean value that controls whether to execute the property filter. The structure switcher is a Boolean value that indicates the target is in a single or a sequential structure. In our experience, such a simple input is a non-negligible advantage comparing with existing countermeasures that need pre-processing because the following requirements.

- Arbitrary target choosing. Extraction tasks often need to be customised according to the needs of users. However it is almost impossible to induce a set of highly accurate rules that suit all pages in advance because users may choose any part of any page of any site with specific application requirement. Even within the same page, different users may choose different targets.
- Timely response. Normal data users may have limited programming skills and focus on the application logic more than technique procedures. It is impractical to require them to quickly design a decent pattern (e.g., HTML subtree) to match the target data in page variants correctly. For professional users who could recover the pattern and tune parameters well, due to the large-scale data operation, each repair work still becomes time costly.
Stable web scraping

- **Only HTML files.** The extractor should adapt to mainstream data rich webpages. Using only HTML files will reduce the bandwidth overhead and reduce the failure of the entire extraction process due to unsuccessful acquisition of other ancillary files like CSS or JavaScript files.

- **Compatibility.** The extractor should be compatible with existing techniques such as XPath. It works like the plug and play style which can be easily combined with other methods.

7 Related work

The main objective of web scraping is different from the web crawler (Zhao et al., 2016), information retrieval (IR) systems (Behnert and Lewandowski, 2017; Tulasi et al., 2017) and machine learning/NLP-based information extraction (IE) technologies (Mintz et al., 2009; Wu and Weld, 2010; Ritter et al., 2013). They focused on either gathering/querying whole webpage documents or extracting logical contents from human language texts, and thus are out of our scope. As the requirements discussed in Section 6.5.3, according to the chosen manner of the target data, we classify existing scraping methods into two categories: fixed-feature-target extraction and arbitrary-specified-target extraction.

Fixed-feature-target extraction automatically detects and extracts target information which usually has a fixed format or topic. The extraction rules are either predefined or induced by a set of samples. For scraping the deep web data, Gatterbauer et al. (2007) and Furche et al. (2012) proposed techniques to recognise the input query element such as the `<form>` and then extract the response messages. Similarly, Chen and Cafarella (2013) focused on extracting data from the tabular environment within a page. Audeh et al. (2017) developed an open source scraper to extract the structured data from web forums and represent them as semantic structures. Some studies (Liu et al., 2010; Fang et al., 2018; Figueiredo et al., 2017; Garcia et al., 2017) aimed to automatically locate the main data region from webpages and then segment all structured data records from that region. Extracting specific contents like main contents of news articles or comments of products from various web sources is also a field with active developments (Reis et al., 2004; Han and Tokuda, 2009; Wu et al., 2015). Some of these methods use visual features, (e.g., type, font colour, screen coordinates) to locate webpage elements. Sánchez et al. (2016) proposed an advanced method that extracts communities of users having similar opinions for a given topic in the Twitter platform. It used the stream API of Twitter to collect all the tweets that contain specific keywords.

Arbitrary-specified-target extraction is more applied to vertical wrappers. Users first specify the target data in a page and look forward to automatically extract them from page variants. Our approach belongs to this type. Kowalkiewicz et al. (2006) and Han and Tokuda (2008) attempted to generate relative paths that could robustly represent the location of designated data in the page, while they cannot deal well with the future page changes (Dalvi et al., 2009). Dalvi et al. (2009) presented an algorithm that trains change models to generate a list of XPaths by probabilistic ranking. Cohen et al. (2015) proposed a supervised algorithm that first extracts contextual tree structure from training samples and then performs a recursive tree matching search to extract the target data from page variants. Omari et al. (2017) constructed extractors that adjust their precision dynamically to handle page changes without sacrificing precision on the training set.
These approaches need a careful pattern design, or a training phase conducted by human experts and thus are not suitable for the scenario where the flexibility for target choosing is required. Ferrara and Baumgartner (2011) proposed an algorithm improved from the tree edit distance algorithm, which invented a weight function addressing less importance to slight changes during the sub-tree matching. This method works well for page variants with minor structure changes. Leotta et al. (2016) proposed an algorithm that generates robust web testing-oriented XPath for automated web application testing. This work mostly focuses on the UI components oriented extraction such as the input button in a form field. It ranks the robustness of element attributes based on some heuristic rules during the generation of the XPath. \textit{OXPath} (Furche et al., 2013) extended the XPath with more semantic actions (e.g., click, form filling) and markers. Benefit from more machine-readable it improved the robustness and in the meantime it also lost some compatibility and requires higher learning costs.

In addition to the above methods that focus on scraping data from HTML documents, in recent years, some researchers aim to mining information from image and video files for webpage categorization/classification (López-Sánchez et al., 2017b,a). These methods also need a training process conducted by human experts. Stanton et al. (2017) developed an interface that enables the user to acquire immersive information by instantly switching between the 2D hypertext interface and an 3D environment that incorporates 2D HTML elements.

8 Conclusions

In this paper, we proposed a web scraping approach that focuses on the stability problem caused by page variants. We defined the concepts of data type and node distance, which describe the characteristics of page elements and the layout relationships between nodes in HTML trees. Our approach consists of an algorithm that searches for the approximate location, i.e., neighbour zone, of the desired information and an original method that measures the similarity of page elements based on information carried by their XPaths. Experiments on a large set of real-world webpages show that our method has better stability for web scraping compared to the XPath method (F1-score increased 0.118 in \textit{Dataset1}, 0.891 in \textit{Dataset2}). By combining the neighbour zone (SSN) and the path similarity (SSP) components, our method, i.e. SSNP, get a more stable extraction result (0.871 vs 0.95 vs 0.961 in \textit{Dataset1}, 0.644 vs 0.636 vs 0.891 in \textit{Dataset2}). Our approach supports arbitrary target data selection in a page and returns the extraction results in time, since it only needs the HTML source files and does not require pre-processing such as labelling samples or designing patterns. This is significant because it complements existing XPath-based methods and is flexible for customised extraction.

In our future work, we think the following directions are important and interesting. We will further develop the calculation of unchanged node pairs and the semantic similarity of attributes to further improve the effectiveness of the SSN and SSNP method. For example, we may optimise the mapping for HTML trees by developing a cost function that specifies a higher cost to the change operation in the TED algorithm. We will construct a customisable scraping platform, which supports target choice through a GUI and offers various extraction methods as plugins in the backend. In
addition, from the perspective of protecting information privacy and intellectual property of websites, anti-extraction techniques are a crucial extension.

Acknowledgements

We gratefully acknowledge the helpful advice from Takehiro Tokuda (professor emeritus, Tokyo Institute of Technology, Japan), Chao Liu (Tuniu Corporation, China) and Chao Wu (Nanjing Tech University, China).

References


