論文 / 著書情報
Article / Book Information

題目(和文)	
Title(English)	Stable Extraction Methods for Partial Information on the Web
著者(和文)	高鵬
Author(English)	Peng Gao
出典(和文)	学位:博士(学術), 学位授与機関:東京工業大学, 報告番号:甲第11026号, 授与年月日:2019年2月28日, 学位の種別:課程博士, 審査員:佐伯 元司,渡部 卓雄,権藤 克彦,西崎 真也,小林 隆志
Citation(English)	Degree:Doctor (Academic), Conferring organization: Tokyo Institute of Technology, Report number:甲第11026号, Conferred date:2019/2/28, Degree Type:Course doctor, Examiner:,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

Stable Extraction Methods for Partial Information on the Web

Peng Gao

March 2018

Department of Computer Science

Tokyo Institute of Technology

Contents

List of Figures

List of Tables

Abstract

Extracting the designated information from the vast amounts of web pages on the Internet becomes a prerequisite for web data analysis. Partial information extraction techniques based on XPath enable users to consistently extract information of interest from web pages that do not provide a structured interface. However, XPath-based extraction is likely to fail when encountering page variants caused by inconsistent template or internal structure changing over time, resulting in a high cost of repair. Countermeasures based on pattern matching or model learning often require a time-costly preprocessing, which is not suitable for cases where the target data is frequently re-designated. In this dissertation, we present two new extraction methods for the stable scraping of arbitrary designated data from web pages. The first method, namely neighbor zone based method, determines the required information in the changed page based on the unchanged elements and layout relationship in their HTML trees. The second method, namely path similarity based method, searches the information of interest by ranking the similarity of the characteristic of page elements based on their XPath information. We also combine these two methods together to get a higher stable hybrid method. Experiments on a large set of real-world web pages show that our methods have better stability for web scraping, compared with the XPath-based extraction. With the hybrid method, in the two datasets, the F1-score increased by a maximum of 0.119 and 0.891 respectively. Our methods only need HTML source files and require no preprocessing like manual pattern designing or model learning, which allows users to flexibly extract information of interest from large-scale web pages.

Acknowledgements

I am deeply grateful to many people for supporting this work. I would like to express my gratitude to Professor Emeritus Takehiro Tokuda and Professor Motoshi Saeki of Tokyo Institute of Technology for supervising this work and dissertation. I have greatly appreciated their continuous support and valuable advice. I would like to thank Assist. Prof. Tomoya Noro and all members of Lab Tokuda for their helps and supports. I received a lot of insightful comments and suggestions that became great help to improve my work. My senior, Dr. Hao Han, gave me helpful comments and encouragements.

This work would not have been possible without support from my family. I am deeply grateful to my wife Chao Wu, who gave me generous support and warm encouragement to complete my work. I would to extend my gratitude to my parents. This dissertation would not be completed without all of the longtime support they have given to me, therefore this dissertation is dedicated to my family.

Chapter 1

Introduction

1.1 Background

In the late 1980s, the World Wide Web (aka."WWW", "the web") implemented the "largest library" of the world through interlinked hypertext documents. Since then, people can acquire information and generate personalized knowledge via the internet quickly. With the popularity of data-rich web applications in recent years, such as social networking, online shopping, mobile browsing, etc., the internet provides access to extraordinary large amount of information through web pages. These web pages naturally consist of heterogeneous information, for instance, textual contents, hyperlinks, visual images or other multimedia data[1], which are continuously produced and consumed online by various platforms. As illustrated in Figure 1.1, people expect an intelligent agent that can help to automatically extract the information of interest from web pages for further usage. Accordingly, partially scraping the required information from web pages plays a key role of precondition on the web data analysis and web application hybrid (mashup) technologies[2].

Partial information extraction (PIE) is also known as web scraping, web data extraction, web harvesting or wrapper, which is a process of automatically extracting the required data from publicly available web pages. A typical process of the PIE application can be generally addressed as the following four steps.

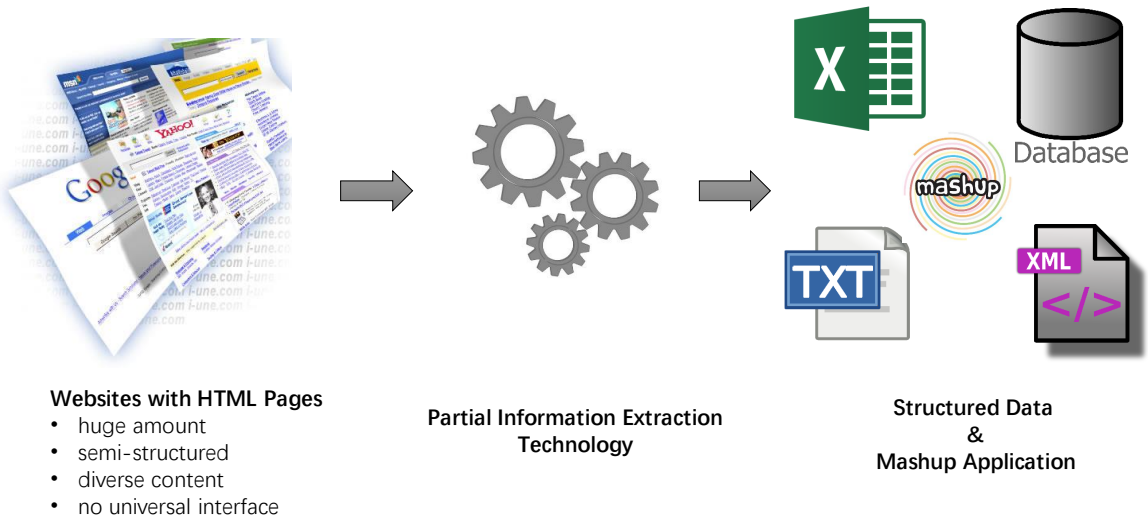


Figure 1.1: **Partial Information Extraction on the Web**

(1). Users select a web page and designate a target data in the page.

(2). Users locate the target data through a locator, e.g., XPath, regular expression and etc.

(3). Users extract the designated target node using the client-side locator.

(4). Optionally, users can integrate the extracted data with other parts to realize original content or service, e.g., a Mashup application.

Many organizations and individuals have paid great efforts to collect data in web pages at a very large scale for obtaining scientific insights. For instance, researchers trace digital contents generated by users of social network website *Facebook* for analyzing human behavior[3]; the statistician accurately predicted the results of presidential elections based on analyzing data that scraped from the internet[4]; comparison shopping sites like *manmanbuy.com* 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) [5, 6, 7, 8] on it, which specifies how to traverse the trees. The black solid line in Figure 1.2 shows an HTML tree of a camera detail page simplified from an online shopping site. To extract the price value carried by the text node with label ⑥, we can use the following XPath expressions,

- P_1 = /html/body/div[2]/div[2]/table/tr[1]/td[2]/text()
- P_2 = //div[@id='price']/*/tr[1]/td[2]/text()

where P_1 is an absolute XPath that fully describes the path starting from the root node html to the leaf node text containing the desired information (¥8,699); P_2 is a relative XPath using a double slash which can start from any intermediate node of the tree depending on user's specification.

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

1.2 Motivation

Consider a scenario in which a owner of a personal e-commerce site, who searches and recommends exquisite cross-border products for his/her local circles of friends. Every day the owner monitors and scraps latest products' detail information such as price, reviews from various shopping sites, analyzes the cost-effective and then updates the desired information into the site. The whole process largely depends on the result of the web scraping, in case which fails it becomes very laborious and costs significant. However, the scraping process based on XPath suffers from the *stability problem* which makes the path break and the extraction fail frequently[9]. On the one hand, websites use similar but inconsistent templates to construct even pages of the same category. Our previous work[10] showed that the top page of *Yahoo!*

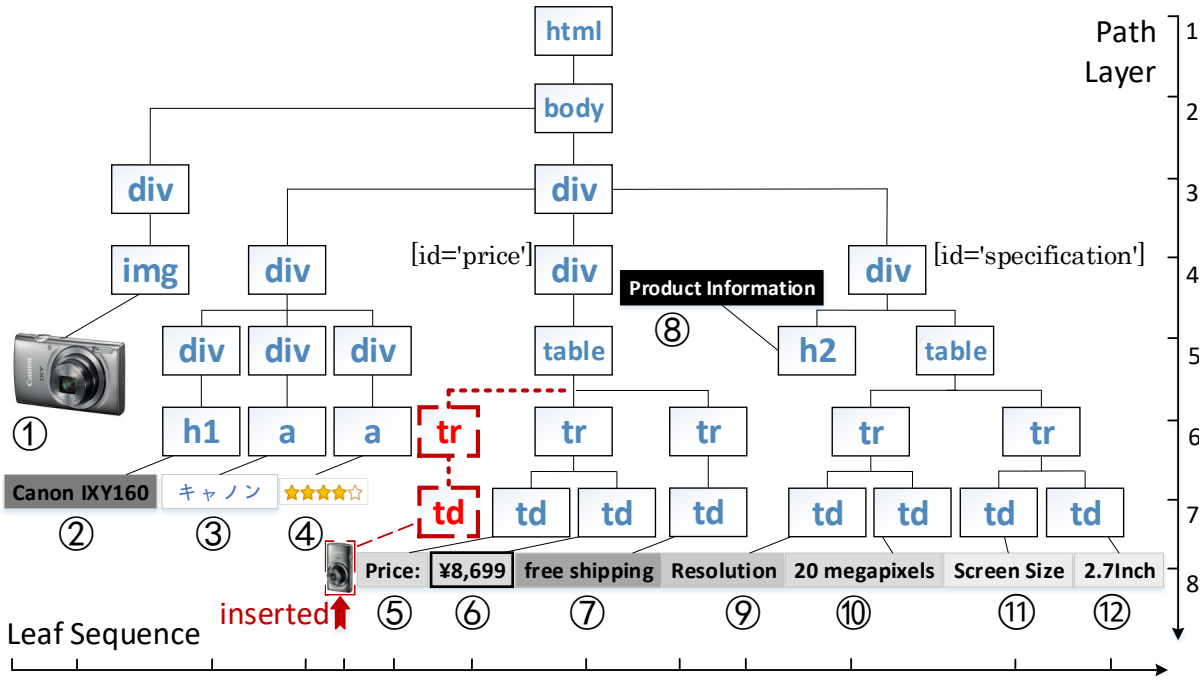


Figure 1.2: **Sample HTML Trees of a Web Page and a Page Variant**

A web page and its page variant are mixed in Figure 1.2. The page variant just inserts red dotted branch marked as *inserted* in the page and other nodes are unchanged.

News and *BBC Country Profiles* 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 web page may change at any time without notification because the layout is updated. According to the literature[11], 6.5% of a sample of 55,000 HTML pages changed every minute, while 41.6% changed every hour. I refer to such a web page as a *page variant* where the initial XPath fails to repeatedly extract the designated information, which is caused by inconsistent template or internal structure changing over time.

As illustrated in Figure 1.3, the stability problem caused by a page variant can be defined as follows. Let T_1, T_2 denote HTML trees of a web page 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 specified 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 red dashed line in Figure 1.2, when a new td element carrying a thumbnail image node with label *inserted* is inserted into the left of the price value ⑥, the extraction for the price using XPath P_1 or P_2 fails, which returns nothing in this case.

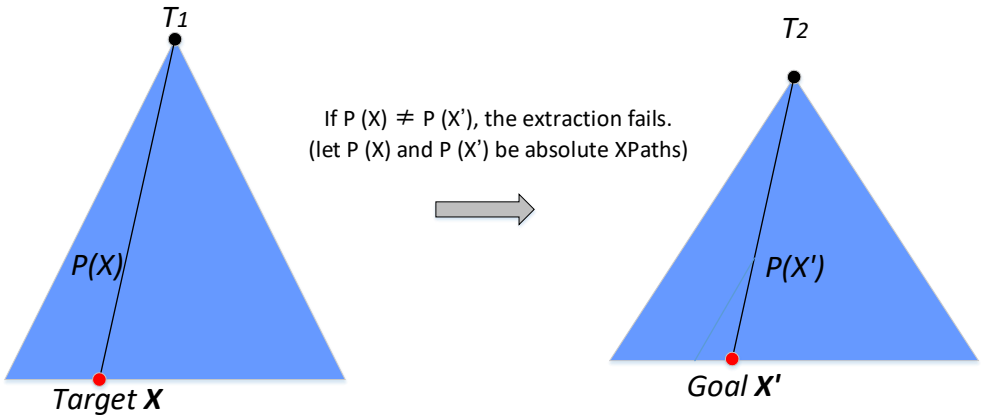


Figure 1.3: **Problem Statement**

1.3 Existing Approaches and Challenges

To address this problem, i.e., reducing manually repairing, many research efforts have been presented.

- Main HTML tree based methods can be classified into two groups: pattern matching and model learning based approaches.
  - The pattern matching based approaches[12, 13, 14, 15, 16] usually learn or design a pattern for the target data, e.g., a subtree or an XPath, and then locate the target data in the page variant by matching the pre-generated pattern.

  - The model learning based approaches[9, 17, 18] construct stable extractor by learning change models from historical versions of the page.

- Vision based methods, including the ones combined with the tree based methods e.g.,[19, 20, 21, 22], use visual features, (e.g., type, font color, screen coordinates) to locate web page elements. This kind of approach more focuses on extracting specific target contents with a larger visual area, such as data record, web news article, page segmentation and so on. When the target data is a single leaf element, the extractor is fragile to changes of the web page layout.

In order to locate the *Goal* automatically, these countermeasures either require a beforehand pattern design, or need to manually label features on a number of sample pages for training model and tuning parameters in advance. However, the complex pre-process is not efficient in case the target data is changed frequently. Moreover, sometimes it requires manual re-labeling if the extraction failed and is also unfriendly to normal data users with limited programming skills. Besides requiring carefully designed preprocessing, another challenging issue is that it is difficult to locate the *Goal* in a page variant directly through only the HTML source file, which offers limited layout features, with no help from subsidiary files like cascading style sheets(CSS).

There are also some techniques that focus on generating robust extraction locator directly, e.g., XPath, regular expression, subtree, while some of them[23, 24, 10, 25] cannot deal well with the future page changes[9]; some of them need supervised learning[26, 27]; and some of them have different scope from ours[21, 28, 29].

However, few of them are suitable for the above application scenario: a customizable web data scraper. The target data may frequently be re-designated by users. For example, different data from multiple web pages are obtained by an end-user mashup tool [30] to be immediately presented to users. Therefore, I aim to make the scraper without pre-processing steps and as stable as possible. The specific challenge I address is that with no help from:

(1) a pre-processing like model learning or pattern designing, or (2) visual subsidiary files like Cascading Style Sheets (CSS), searching the arbitrary specified data in page variants automatically is difficult. In our work, we focus on the HTML tree based methods because it is well adopted in practice[29]. Our goal is to develop extraction methods surpassing the XPath in terms of stability, which also can be flexibly used for different application scenarios. We discuss the above challenging issues in Section 7.7 and analyze the related techniques in Chapter 2.

1.4 Our Approaches and Contributions

As addressed in Section 1.3, for instance, besides requiring carefully designed preprocessing, it is difficult directly to locate the Goal in a page variant through only the HTML source file which offers limited layout features. 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 web pages, we find that the high similarity of following features of the *Target* in T_1 and *Goal* in T_2 helps people to identify it correctly.

- *Layout relationship* (e.g., a distance between locations, neighbor information)
- *Element characteristic* (e.g., HTML element type, semantic information)

Following with the shopping site's example in Figure 1.2, although an image node is inserted (red branch), we easily found the price value because it is very similar to the page before inserting the picture where it is a numeric value just following the text node "price" and its position is in the middle area of the horizontally arranged leaf nodes.

Inspired by these observation, i.e., similarity of layout relationship and element characteristic, we present two new approaches in this thesis. The first method uses the layout relationship to determine the area of the desired information, namely *neighbor zone*, which consists of a

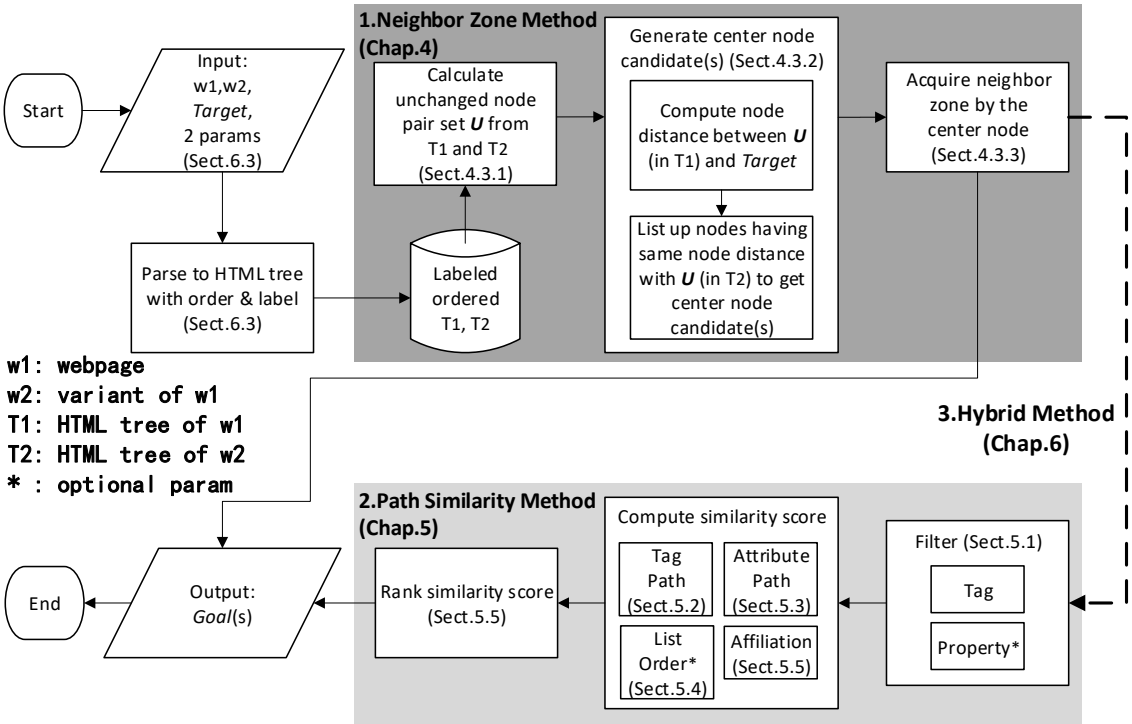


Figure 1.4: **Workflow of the Extraction Approaches**

fragment of leaf nodes where the central one ,namely *center node*, is deemed as the *Goal*. The second method searches the most likely node as the *Goal* by ranking the *path similarity* based upon the characteristics derived from their XPath information.

Furthermore, based on the observation of the experiment results, we combine these two methods to get a higher stable hybrid solution, by letting the output of the first method based on the neighbor zone as the input of the second method based on the path similarity. We use the Figure 1.4 to depict the main components and workflow of the hybrid method, as a matter of course, which covers the two proposed methods.

The main contribution of our work is that we designed and implemented two highly stable extraction methods and a hybrid method of them that support arbitrary target data selection in a web page. They only use HTML source files and need no pre-processing, which makes

them be compatible with existing XPath-based methods and flexible for customised large-scale extraction. The experiments on two complementary datasets, containing more than 30,000 target data selected from 102 real-world popular websites, show that our approaches' advantage over the XPath in terms of extraction stability. To be specific, the originality of our work are the following aspects:

- It develops a distance measurement between a node and a leaf node in the HTML tree.
- It proposes an algorithm 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.

1.5 Organization of the Thesis

The rest of this thesis is organized as follows. Chapter 2 discusses related work of this field. The preliminary knowledeges are given in Chapter 3. The method based on the *neighbor zone* is addressed in Chapter 4. The method based on the similarity of page elements is presented in Chapter 5. The hybrid mechanism and implementation environment of above two methods are presented in Chapter 6. In Chapter 7 we conduct experiments and evaluation to show the effectiveness of our proposals. Chapter 8 concludes the thesis and present possible future work.

Chapter 2

Related Work

The main objective of partial information extraction is different from the web crawler[31, 32, 33], information retrieval (IR) [34, 35] and machine learning(ML)/natural language processing(NLP) based information extraction (IE) technologies[36, 37, 38]. They focused on either gathering / querying whole web page documents or extracting logical contents from human language texts, and thus are out of the scope of this thesis.

Beyond the inchoate manual-based approaches[39, 40, 41], in this thesis, we focus on automatic web scraping technologies, which are also compatible with the existing web environment. A lot of research efforts have been addressed to automatic web scraping while very few of them directly address building more stable mechanism. We believe the stability is the backbone of partial information extraction technology which deeply affects the development of modern extraction techniques.

In this chapter, we first introduce the state-of-the-art extraction technologies and then, according to the target choosing manner mentioned in Section 1.3, we classify existing scraping methods into two categories: fixed-feature-target extraction and arbitrary-specified-target extraction.

2.1 State-of-the-art Solutions

Two state-of-the-art solutions have been presented to handle the stability problem from the ground up[2], both of which focus on letting computer "understand" what contents web page contains. The first solution is the XML-based web document approach. This approach divides each HTML document into two XML documents: an user-defined content-oriented XML format file and a XSLT-like (Extensible Stylesheet Language Transformations) program file, where the latter file transforms the former file to a form-oriented format e.g., XHTML (Extensible HyperText Markup Language) format. Accordingly, extracting data from such kind of web page will be equal to extracting the data from the former file whose format is well-defined in the latter file, e.g., XSLT, which makes it likely to be stable over time. The second solution is the semantic web approach. In this situation, HTML documents would have been attached with meta-information to describe the semantics of their content. The semantics is described in a a metadata data model, e.g., Resource Description Framework (RDF), which is linked from the HTML document. The meta-information links an HTML element to an entity in the RDF document, which describes the content of the HTML elements and then the computer could understand the RDF document.

However, both of these solutions are still not likely to be universally adopted for partial information extraction due to some technical and social issues. Firstly, there are a huge amount of existing web pages that need to be re-designed and updated, while even if we achieved that, such kind of pages may still face tons of compatibility problems with existing well-formed systems. The semantic web technology also is challenged by uncertainty and inconsistency problems. As for social issues, many web content authors do not want to offer the information in a structured format that can be directly understood by computers. For instance, Google Ads wants people to access web pages to earn profit, therefore, they could develop obfuscation modules to prevent automatic data extraction.

2.2 Fixed-feature-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 example, for scraping deep web data, researchers proposed techniques to recognize the input query element such as the <form> and then extract the response messages[42, 43]. Similarly, in [44, 45], researchers focused on extracting data from the tabular environment within a page. We introduce these techniques in the following subsections respectively.

2.2.1 Form/Table Extraction

The HTML form in a web page enables users to input query text, e.g., hotel reservation, stock prices, in the input element which usually displayed as a input box in the front end. Then the function in the back end of the form is going to query information from the remote databases. Such kind of data is called deep web [46] which is very important for search engines. The approaches proposed by Furche et al.[47, 48], firstly search for a <field> element enclosing the input label, if fails, then search for labels in the enclosing <form> element. Finally if it still fails, they fall back to find a text field, that layout wise is in the proximity of the form element. Similarly, in [44, 49, 45] authors focused on extracting the table information within a page. The table extraction task is challenged by the fact that instead of carrying information, the <table> element is more used for well arranging the layout of the web page other than exhibiting tabular information.

2.2.2 Data Record Extraction

Some studies[19, 20, 50, 22, 51, 52] aimed to automatically extract all structured data records from that region. Most of these methods are based on the HTML tree alignment or vision layout tree matching approaches. Their main objective is to recognize and segment the data record region from web pages. These approaches mainly consist of three steps, (1)

detecting the main area of data record in the web page; (2) identifying the boundary of each individual record and then segmenting them; and (3) aligning and extracting data items from the identified data records.

2.2.3 Specific Content Extraction

Extracting specific contents, e.g., main contents of news articles or comments of products from various web sources, is also a field with active developments[26, 53, 54, 55, 56, 57]. Reis et.al [26], proposed a domain-oriented method to automatically extract news from normal websites with no human intervention. The approach firstly classifies web pages into news pages and not news pages, then searches news part of each news page and extracts their main components. The *AkwanClipping*[1] is a commercial Web news extracting system that fetches daily news from main Brazilian newspapers. Han et.al [53] presented a relevance-based analysis method to extract the news article contents from the general news pages without the analysis of news page layouts before extraction. The main advantage of this method is that it has high performance at parsing and analyzing page's structure and does not need a time-consuming learning procedure. In [58], an open source scraper was developed to extract the structured data from web forums and represent them as semantic structures. [59] proposed a 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.

2.2.4 Pros and Cons

In summary, the pros and cons of the fixed-feature-target extraction are summarized as follows.

- Advantage
  - They accurately, efficiently extract specific data for large scale web extraction.

[1]http://www.akwan.com

  - The user does not need to designate target every time.

- Disadvantage
  - It is expensive to change the extraction rules or switch domain.
  - The desired/target information is fixed (cannot choose).

2.3 Arbitrary-specified-target Extraction

Arbitrary specified target extraction approaches are more applied to vertical wrappers. Users first specify the target data within a web page and look forward to repeatedly extracting them. Our approaches belong to this manner. We divide related researches of this manner into two groups: path-based approaches and pattern-matching / model-learning based approaches.

2.3.1 Path-based Extraction

Taking advantage of the fact that a great amount of and still increasing web pages are rendered by template system, many approaches have been proposed to analyze the structure of web pages with the purpose of manual or semi-automatic example-based data extraction.

ANDES[6] is a XML-based methodology to use the manually created XSLT processors to realize the data extraction from web pages. Similarly, Marmite[60], implemented as a Firefox plugin using JavaScript and XUL, uses a basic screen-scraping operator to extract the content from web pages and integrate them with other data sources. The operator uses a simple XPath pattern matcher and the data is processed in a manner similar to Unix pipes. PSO[61] is an approach to extract partial parts of web pages. It keeps the view information of the extracted parts by using the designated paths of tree structures of HTML documents. Crunch[62] is a HTML tag filter that retrieves content from DOM trees of web pages after analyzing their HTML documents. These extraction tools face the stability problem as we mentioned above. When websites update the pages by modifying templates irregularly, the extraction process

would fail because paths are no longer available or wrong results are extracted. The extraction mechanism employed by these methods makes them difficult or even impossible to re-generate suitable paths automatically.

Dapper[63] is a screen scraping tool that allows users to extract partial information from web pages. It has a powerful graphical user interface (GUI) that allows users to scrape data from the page without programming or locators like regular expression and XPath. However, it is not very intelligent to recognize a long list. For a list, users may have to select each individual item. Moreover, users have to provide sample web pages for analysis and execute selection separation many times for enhancing precision, which still could not keep extraction procedure steady after websites update pages' layout.

WIKE[64] and *Web clipper*[65] are web authoring environments that enable end-user to dynamically extract information from various web resources. The system provide users an environment to extract content and construct a personalized web page while which makes it weak than now popular UI-component based mashup applicaitons. An advantage of them is that they enforce a robust mechanism based on multiple extraction patterns defined in advance. The tunning process carried out manually would be tedious if users want to get a ideal extraction result. For each component from web page, the Web clipper need to do at least once the training process even for the pages that generated by very similar templates. As the number of extraction patterns increasing, the training process will become laborious.

Several approaches attempt to evaluate robust extraction locator directly, e.g., XPath, regular expression, subtree, while some of them[23, 24, 10, 25, 66] cannot deal well with the future page changes[9]; some of them need supervised learning[15, 27]; and some of them have different scope from ours[26, 21, 28]. The *Robular*/*Robular*+ [67, 29] proposed algorithms that generates robust XPath expression for automated web application testing. They rank the robustness of element attributes based on some heuristic rules during the generation of the XPath. The robular method depends on the exact matching of HTML attributes in the path, e.g., the value of the `class` attribute. Our experimental results show that these two methods are

more suitable for web application testing that focuses on the UI components oriented extraction such as the input button in a form field, however, for extracting tasks that focus on content-oriented data in the page, their effectivenesses are not better than XPath. The *OXPath*[28] 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.

2.3.1.1 Pros and Cons

In summary, the pros and cons of the path-based extraction approaches are presented as follows.

- Advantage
  - Flexibly changing target
  - Quickly response for transferring result
  - Light-weight and easy implementation
- Disadvantage
  - They cannot deal well with the future page changes.
  - The specialized scraping language, e.g.,OXPath, is powerful but has problems of compatibility and learning cost because the extraction expression needs to be inputted by the user.

2.3.2 Pattern Matching / Model Learning based Extraction

The *regular expression* technology provides a concise mean for matching strings of text, such as particular characters or patterns of characters. A regular expression is written in a formal language that can be interpreted by a regular expression processor. Then the resulting

pattern can be used to create a matcher object which can match arbitrary character sequences corresponding to the regular expression. Regular expression runs more flexibly with various logical, intersection, and union operators. However, generating a set of regular expressions [68] costs more time than XPath. And if websites update the layout of web pages, the regular expression also faces the stability problem where the re-generation requires a very careful manual repair process handled by human experts.

The *string matching* and *tree matching* algorithms are employed to match the HTML trees. The string matching methods treat web page as a linear string architecture. The HTML tree matching methods parse the web page into a tree structure. Despite the inherent limitations of the algorithm which will be introducted in Section 3.3, this kind of approach is widely adopted for tyhe web data extraction systems[69, 70, 71, 72, 73].

Machine Learning techniques[74, 75, 76, 77, 78, 79, 80, 81, 82] are suit well to extract specific domain information from web pages. They require a training phase in which domain experts provide some manually labeled sample pages. It is important that they need preparing sample pages from the same domain but in different structures. The reason is that even in the same domain, different page templates are used to generate dynamic contents, and thus, the extraction system should have the ability to learn how to extract information in these similariy templates. In the following we shortly describe some Web Data Extraction approaches relying on Machine Learning algorithms

In [13], researchers proposed an algorithm derived from the tree edit algorithm, which uses a special weight function for target pattern matching. In [9, 17], authors presented an algorithm that trains change models to generate a list of XPaths by probabilistic ranking. The [15] 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. In [27], researchers construct extractors that automatically adjust their precision dynamically to handle page changes and do not sacrifice precision on the training set.

2.3.2.1 Pros and Cons

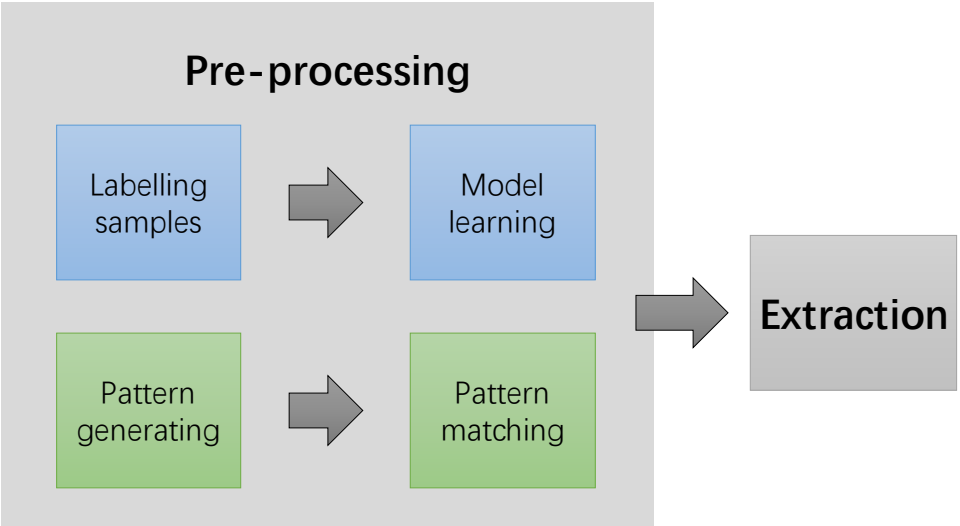


Figure 2.1: **Workflow of the Pattern Matching / Model Learning based Extraction Methods**

The advantage of pattern matching / model learning based extraction are presented as follows.

- Advantage
  - The stability performance is higher than other approaches.
  - The desired/target information can be choosed flexibly.
- Disadvantage
  - As illustrated in Figure 2.1, a preprocessing phase like pattern design, model learning is needed to be conducted by human experts. Accordingly, it causes a high manual involvement (e.g. sample labeling, pattern designing) and thus cannot offer the extracting result to users in time. Therefore they are not suitable for the scenario where flexibility for the target choosing are required.

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 web page categorization/classification [83, 84]. These methods also need a training process conducted by human experts. In [85], researchers 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.

Chapter 3

Preliminaries

In this chapter, we present some preliminary knowledges related to the stable partial information extraction technique.

3.1 Web Page

A web page is a computer document that is transmitted on the web. Most of web pages are created by a standard markup language: Hypertext Markup Language (HTML). Web pages are usually displayed by the web browser on a desktop monitor or mobile devices. Web browsers coordinate various web resource elements for the written web page, such as style sheets, e.g., Cascading Style Sheets(CSS), scripts, e.g., JavaScript, to present contents of a web page. Furtherly, web pages provide hyperlinks, often referred to as links, to "jump" to other pages.

As shown in Figure 3.1, the web standards model consists of HTML, CSS and JavaScript. The HTML controls the page elements and contents of a web page, such as headings, paragraphs, tables, bulleted lists etc. All text, links, including links to the images displayed on a web page, are in the HTML.

CSS controls over the formatting and layout of the web page such as the background colors, font sizes, borders, etc. CSS works on a system of descriptors, which can select and set values for different properties of the page elements. In early phase, the style specifications are also

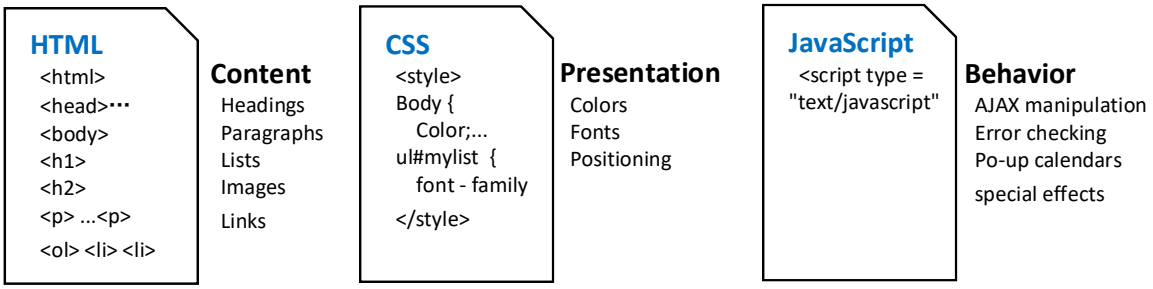


Figure 3.1: **Web Standards Model: HTML CSS and JavaScript**

part of HTML, which makes HTML very difficult to be read, and thus they are separated. An example CSS rule is shonw as the following code. The content enclosed in   tags will be colored blue and have the double line height.

```
p {
    line-height: 2;
    color: blue;
 }
```

JavaScript is the scripting language used to control behaviors of a web page. For example, it can be used to validate the input password of your email like checking if it contains illegal character or not. Anything from animating page elements such as menus, input functionality and so on is handled by Javascript. Even the functionality is very different, just like CSS, most mordern JavaScript enforces the function on a target HTML element.

On the Internet, a web browser receives HTML documents from remote web server and render them into multimedia web pages. Figure 3.2 shows such kind of Web Request-Response protocol in the client-server computing model. Within this environment, an user submits an Hypertext Transfer Protocol (HTTP) request message to the server. The server, which stores web resources like HTML documents, will return a response message to the user. The response message contains the status information about the request and the requested contents. As

mentioned, the process of putting all pieces (e.g., CSS, JavaScript) of a web page together for presentation is called rendering, which will involve multiple HTTP requests.

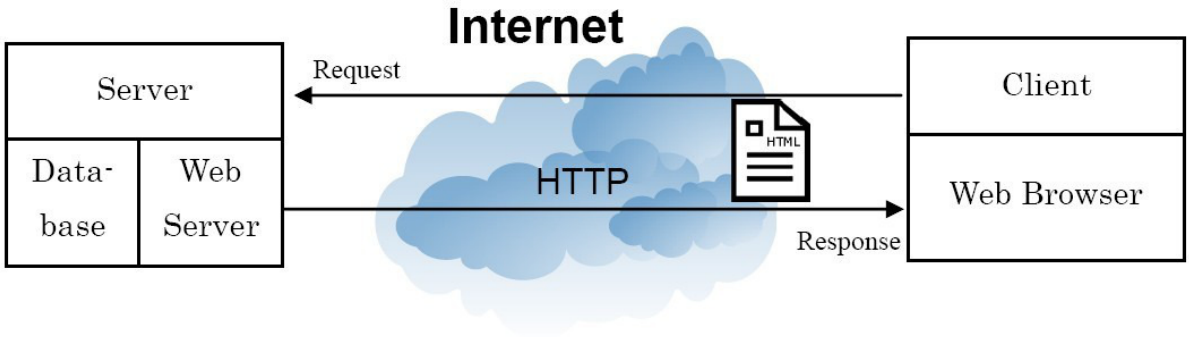


Figure 3.2: **Web Request-Response Architecture**

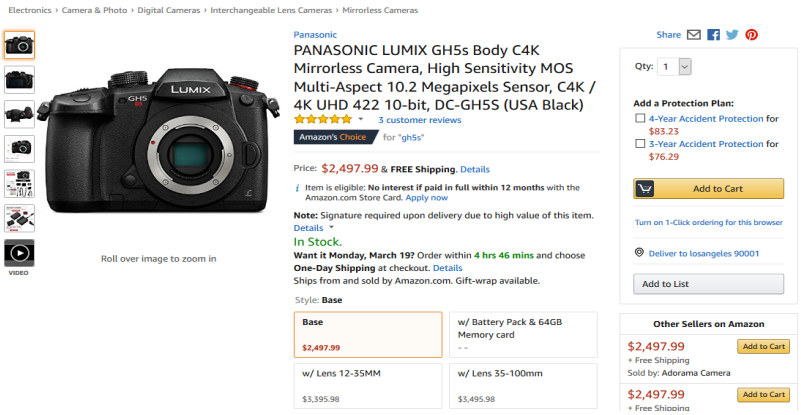
3.1.1 Two Types of Data Rich Web Page

For futher presenting our methods, we introduce the real pages that contain the target data that are wanted to be extracted. As shown in Figure 4.3, mainly there are two types of data rich web pages we concern. The data in these pages are usually retrieved from the back-end database and displayed on the web pages by some fixed templates.

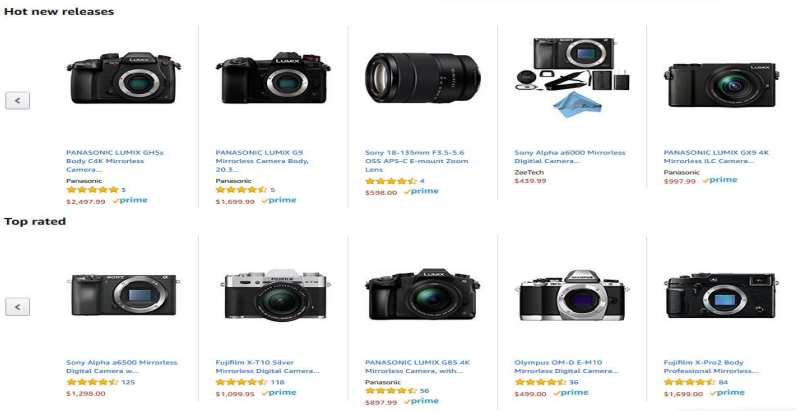
Detail Page Detail pages focus on presenting a single object. Figure 3.3(a) shows a detail web page that contains all details of the camera product(e.g., product name, price, customer rating, images, purchasing information).

List Page list pages contains continuous objects. The layout is segmented by data records which are formatted using the same or quite similar template. Figure 3.3(b) is a list page that contains similar sequential objects.

For list pages, usually it can be developed some heuristics to identify and segment the data region, while for detail pages, it is not easy. As best we know, there are no unified approach



(a) An Example of a Detail Page Segment



(b) An Example of a List Page Segment

Figure 3.3: **Examples of Two Types of Data Rich Web Page of *Amazon.com***

tries to deal with the detail page and list page together. We note that when we mention a page is a detail page or list page, it does not mean that the page contains no other information. We just emphasize the main environment of target data belongs to within the page. For example, in Figure 3.3(b), there are some related product's information in each data record.

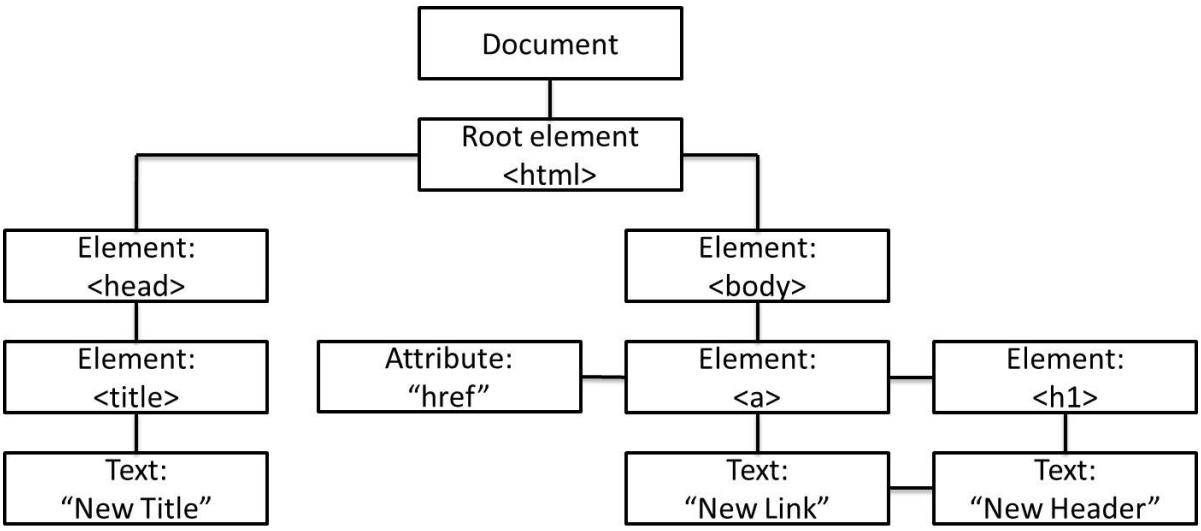


Figure 3.4: **The Document Object Model (DOM) Example**

3.1.2 HTML Tree

When a web page is loaded from the server side to the client side, the web browser constructs a HTML Document Object Model (HTML DOM) for the page, which defines a standard way for accessing and manipulating HTML documents. The DOM is an API (Application Programming Interface) standard developed by World Wide Web Consortium (W3C) for handling "how to get, change, add, or delete HTML elements"[86]. It parses the HTML document as a node tree with elements, attributes, and text. As shown in Figure 3.4, in the node tree structure, the document object is the root node of the HTML document and the "owner" of all other nodes: document node, element nodes, text nodes, attribute nodes, and comment nodes.

Figure 3.5 shows a simple HTML document which is a student&teacher information table block in a web page. HTML uses markup tags (<body>, <table> in the example), which can be nested within each other, to describe web pages and interpret the content of the page. The tags are not displayed on the screen by the browser. Each tag describes an element of the web page. For example, headings, paragraphs, tables, bulleted lists etc., these page elements are used to

denote various types of content in HTML documents, which specifies each piece of content is supposed to be rendered as in the web browser.

```
<HTML>
  <head>
    <title>a simple HTML document</title>
  </head>
  <body>
   <table border=4 width=250 align=center id="teacher_list">
     <caption>Tutor List</caption>
      <tr>
        <th>Name</th>
      </tr>
      <tr align=center>
        <td>Prof. White</td>
      </tr>
   </table>
   <table border=4 width=250 align=center id="student_list">
     <caption>Student List</caption>
      <tr>
        <th>Name</th>
        <th>Gender</th>
        <th>Age</th>
        <th>ID</th>
      </tr>
      <tr align=center>
        <td>Tom</td>
        <td>Male</td>
        <td>18</td>
        <td>001</td>
      </tr>
      <tr align=center>
        <td>Mary</td>
        <td>Female</td>
        <td>19</td>
        <td>002</td>
      </tr>
   </table>
  </body>
</HTML>
```

Figure 3.5: **A Sample of Hypertext Markup Language**

As shown in Figure 3.6, an HTML element usually is defined by a triple: a pair of "start" and "end" tags; *attributes* (some optional and some mandatory) in the start tag; and the content between start and end tags which can be empty or a combination of text strings or HTML elements. Intuitive, the HTML element is everything from the start tag to the end tag such

as <tagname attributes>contents</tagname> For some tags such as br, hr, the end tag can be omitted. The attributes describe additional information of page elements, such as an id for identifying the element, or a location for a hyperlink to jump to.

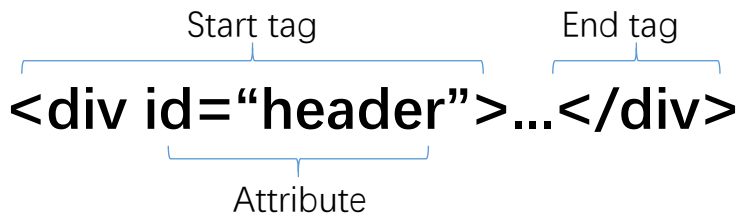


Figure 3.6: **Example of an HTML Element**

The HTML trees of web pages are rooted labeled trees. A rooted tree is a tree with a countable number of nodes and in which the root node is a particular node distinguished from the others. A labeled tree is a finite rooted tree in which each node has attached to it a non-uniqueness label. These labels may be any symbol or other object that one can think of, but usually consist of one letter or symbol from a finite alphabet set, e.g., the set of HTML tags. The node which does not has a parent node is the *root* node. A node which does not has a child node is a *leaf* node. Children nodes of the same node are called siblings.

An edge in a tree is presented as *(child, parent)*. We denote $p(child) = parent$. A node b is an ancestor of node a if and only if $b = p(a)$ or b is an ancestor of $p(a)$. Accordingly a is a *descendant* of b if and only if b is an ancestor of x. A path in tree T is a subtree of T in which each node has at most one child. A subtree S_T of T is a tree that consists of: (a) a subset of all nodes of T : $N(S_T) \subseteq N(T)$ (b) all edges in T that connect these nodes : $E(S_T) \subseteq E(T)$. A forest is a disjoint union of trees. A subforest S_F of a tree T is a graph with nodes $N(S_F) \subseteq N(T)$ and edges $E(S_F) = \{(a, b) \mid (a, b) \in E(T), a \in N(S_T), b \in N(S_T)\}$.

All trees and forests we mentioned are ordered if no special statement is presented. Without the order, when we want to process a tree, it is unclear what the sequence should be, e.g., top

to bottom, or left to right. An ordered tree means a tree in which the sibling order is given, e.g., preorder or postorder (contrarily an unordered tree is a tree with no order among siblings). Many algorithms that make use of trees often traverse the tree in some order, which defines the systematic way they trace each node of a tree. The postorder traversal of an ordered tree is to recursively traverse subtrees rooted in children of the current node from left to right starting from the root node. The postorder traversal of T can be defined by a two-step recursion:

- traverse subtrees rooted in children of the current node (from left to right) in postorder
- visit current node

As shown in Figure 3.7, the nodes of the tree T are $N(T) = \{n1, n2, n3, n4, n5, n6\}$, the edges are $E(T) = \{(n6, n2), (n6, n5), (n2, n1), (n5, n3), (n5, n4)\}$, the label of the node is the alphabet in the figure. The root of T is $r(t) = n1$, $|T| = 6$. The postorder traversal visits the nodes of T in the order as index number of each node. Let $pid(T[i])$ be the postorder id value of the node $T[i]$ and $pid(\mathrm{T}[i]) = i$. We note that the nodes of a postorder tree are ordered as following.

- The $pid(a) > pid(b)$ for any edge $(a, b) \in E(T)$
- For siblings nodes c, d, if $pid(c) < pid(d)$, then c' < d for all descendants c' of c, and c < d' for all descendants d' of d.

Then we can define a node a is to the left (right) of a node b if and only if $pid(a) < pid(b)$ ($pid(a) > pid(b)$) and the node a is not a descendant (ancestor) of node b.

3.2 XML Path Language (XPath)

The XPath (XML Path Language) is initially defined by the W3C as a query language for selecting nodes or node-sets in an XML document. It is based on a tree representation of the XML document, and provides the ability to navigate around the tree, select nodes by a variety of criteria and compute values (e.g., strings, numbers, or Boolean values). Commonly, an

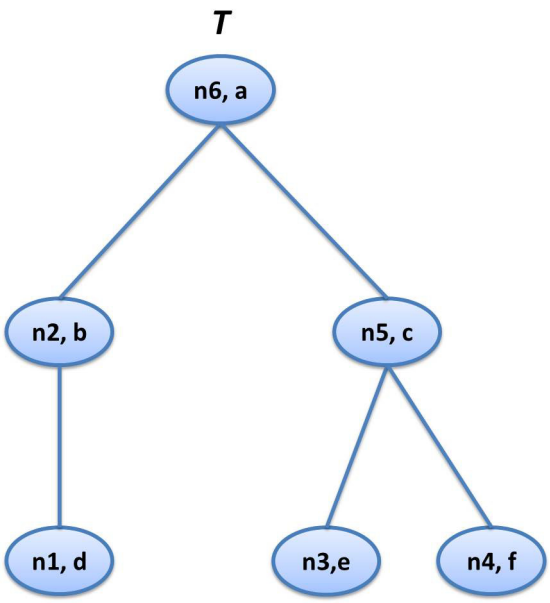


Figure 3.7: **An Example of a Labeled Rooted Postorder Tree**

"XPath expression" is often referred to simply as an "XPath" which intuitively looks very like the path expressions used in the traditional computer file systems.

Since the HTML web page model can be described with a XML-like tree structure, the extraction based on XPath has been widely used. Users can locate and access arbitrary nodes in the HTML tree using XPath. The basis of locating node in the tree by XPath is the *location path*. Depending on whether the root of the tree is the starting point, the location path can be divided into two categories: absolute location path and relative location path.

As the following example, the location path is expressed by connecting basic units called *location steps* with "/".

- Location path = / location step / location step /....

The location step consists of three parts: an axis, a node test (nodetest), and a predicate (predicate). The axis specifies the direction to follow from the starting point, i.e., root node. The XPath has 13 axes such as child, descendant, parent, ancestor. The node test specifies what

kind of node is selected in the direction determined by the axis, and writes a specific element name and keyword. The predicate can narrow down the node set specified by the node test by expressing it with brackets after the node test.

For example, in Figure 3.5, in case we want to locate and extract all names of students, we can use the XPath as follows:

- /html/child :: body/child :: table[2]/child :: tr/child :: td[1]

In XPath, since the keyword representing axes child :: can be omitted, it can be specified like:

- /html/body/table[2]/tr/td[1]

For the relative location path, we can use the following XPath:

- //*[@id="student_list"]/tr/td[1]

The first XPath fully describes path information of target nodes from top node <html> and the second XPath uses node having id attribute as a relative referred node. They can run well in this example page and have their own merits in extraction when the page is changed. For example, if a new table is added over the table Tutor List, the first XPath would lose the effectivity of extraction but the second XPath could still extract successfully. However, if value of attribute id is changed or removed and no new table is added over table Student List, the second XPath could not target the node but the first XPath could keep the right extraction. Therefore XPath based extraction is not sufficient in complex node searching and extraction during long period of time, especially for frequently updated websites.

3.3 Tree Edit Distance

The tree edit distance (TED) between two ordered labeled trees is the minimal-cost sequence of node edit operations that transforms one tree into another[87]. Classically, the set of node edit operations on a tree consists of the following three operations.

deletion delete a node and connect its children to its parent maintaining the order.

insertion insert a node between an existing node and a subsequence of consecutive children of this node.

change change the label of a node.

An example of the operations is shown in Figure 3.8.

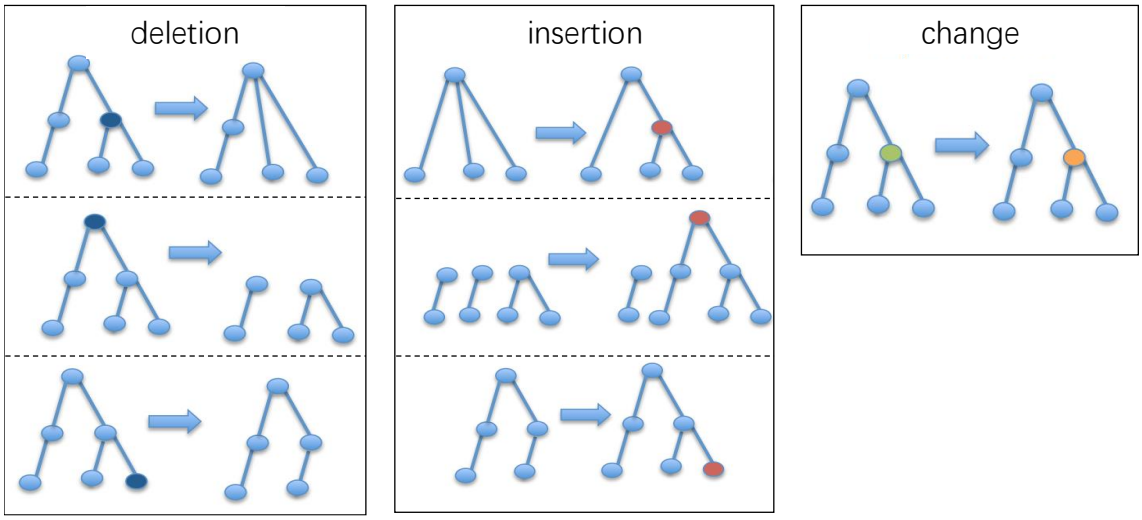


Figure 3.8: **An Example of Elementary Edit Operations**

There are many different sequences that can transform one tree into another. We assign a cost to each edit operation. Then, the cost of an edit sequence is the sum of the costs of its edit operations. Tree edit distance is the sequence with the minimal cost.

Tai et.al [87] proposed the first algorithm to compute the tree edit distance. However, it is complicated and impractical to be implemented. Zhang and Shasha improved it with a dynamic programming approach that reduces the memory space and time complexity by discarding byproduct subproblems[88]. This algorithm 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[12], which usually is much lower compared to the number of all nodes of the HTML. Here we describe Zhang and Shasha's algorithm briefly.

A tree T is a Directed Acyclic Connected (DAC) graph where each node has at most one incoming edge. A $forest$ F is a graph which is composed of one or more trees. Let $D(T_1, T_2)$ be the tree edit distance, γ a cost function of edit operations on nodes and a default blank character Λ. The $\gamma(i, \Lambda)$ denote delete i from F_1; $\gamma(\Lambda, j)$ insert j into F_2; $\gamma(i, j)$ change the label of i to the label of j; $\varnothing$ the empty forest. Let F_1 and F_2 be rooted ordered labeled forests obtained by removing the root of T_1 and T_2; $u \in F_1$, $v \in F_2$ the rightmost node of F_1, F_2 (by postorder) respectively; $F_1 - u$ and $F_2 - v$ the forests obtained by deleting u and v from F_1 and F_2 respectively. The F_1^u and F_1^v denote the subforest rooted in node u of F_1 and the subforest rooted in node v of F_2 respectively, $\varnothing$ the empty forest. Since T_1, T_2 are rooted trees and their root nodes are always the same, the TED $D(T_1, T_2)$ is equal with $D(F_1, F_2)$ which is computed as following dynamic recursion formulas.

$$
\begin{aligned}
&D(\varnothing, \varnothing) = 0 \\
&D(F_1, \varnothing) = D(F_1 - u, \varnothing) + \gamma(u, \Lambda) \\
&D(\varnothing, F_2) = D(\varnothing, F_2 - v) + \gamma(\Lambda, v) \\
&D(F_1, F_2) = min \begin{cases} D(F_1 - u, F_2) + \gamma(u, \Lambda) \\ D(F_1, F_2 - v) + \gamma(\Lambda, v) \\ \begin{cases} \text{if(} F_1 \text{ and } F_2 \text{ are trees)} \\ \quad D(F_1 - u, F_2 - v) + \gamma(u, v) \qquad (*) \\ \text{else} \\ \quad D(F_1^u, F_2^v) + D(F_1 - F_1^u, F_2 - F_2^v) \end{cases} \end{cases}
\end{aligned}
\tag{3.1}
$$

Zhang and Shasha found that the recursion occurs on either the rightmost node of one of the forests that is removed or all but the rightmost tree. Accordingly, the idea of the algorithm is that it always compares the right-most root nodes of the forests. Therefore, if the solution of the subproblems of two trees are already known, we would have a $\mid F_1 \mid \mid F_2 \mid$ dynamic

programming table. So we can maintain a permanent table of size $|T_1||T_2|$ for all subproblems that consist of two trees. Then we only solve each subproblem from the permanent table which is the least costly Tai mapping. The TED algorityhm has the inherent limitations which are that it can not can not match permutation of nodes and it can not map nodes crossing different hierarchical level.

Chapter 4

Neighbor Zone based Extraction Method

In this chapter, we present the stable extraction approach based on the *neighbor zone*. An instinctive observation is that, in page variants, there are some page elements are not or slightly changed, which also have a fix relationship with the desired information. Accordingly the main idea of this method is that instead of directly locating the goal part, we firstly recognize the unchanged parts and then locate the goal part based on the relationship such as the layout relationship (e.g., up or down), element characteristic (e.g., data type, path), etc. For example, in Figure 4.1 both the target and goal part are the first image below the unchanged part.

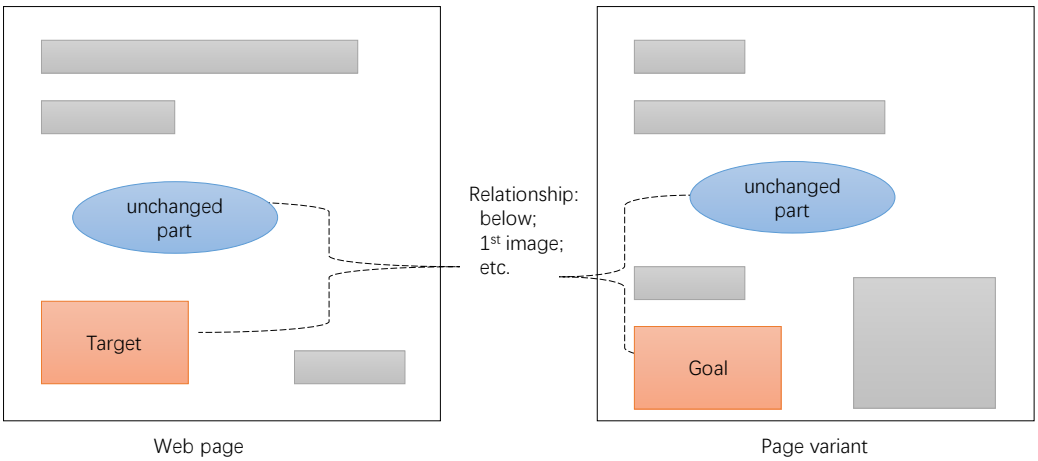


Figure 4.1: **Intuitional Example of the Unchanged Part and Relationship**

The rest of this chapter is organized as follows. An overview of the approach is given in Section 4.1. The concepts of *data type* and *node distance*, which depict the element characteristic and layout relationship, are introduced in Section 4.2. The method for acquiring the neighbor zone is addressed in Section 4.3.

4.1 Overview of Neighbor Zone based Extraction Method

In this section, we give the overview of the extraction approach whose main components and workflow are depicted in Figure 1.4. We use the Figure 4.2 to generally illustrate the concepts and notations of this method. Let T_1 and T_2 denote rooted ordered labeled trees parsed from the web page 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 the tree T in the given ordering. If the context is clear, we use the ordering i as an abbreviated notation to denote the node $T[i]$ in tree T. 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. For the later examples of Figure 1.2 used in this paper, let the node ⑥ (*Target*) is unchanged in T_2. In order to clearly show the correspondence between the nodes in T_1 and the nodes in T_2 without relabeling the node order in T_2, for example, we write them as ⑥ (*Target*) and ⑥' (*Goal*) respectively.

Because directly locating *Goal* in T_2 is difficult, we start from the observations. For the *layout relationship* mentioned in Section 1.4, we use the distances, which measure the layout position between the unchanged nodes and the *Target*, to acquire the *Goal* candidates, which should have similar distance values with the unchanged nodes (Acquire Neighbor Zone in Figure 1.4). We define the *data type* to represent part of the *element characteristic* mentioned in Section 1.4,

More specifically, the layout relationship concerns following two aspects.

- There are some unchanged elements appearing in both web page w_1 and page variant w_2. We refer to the corresponding nodes of the unchanged elements as the *unchanged node*

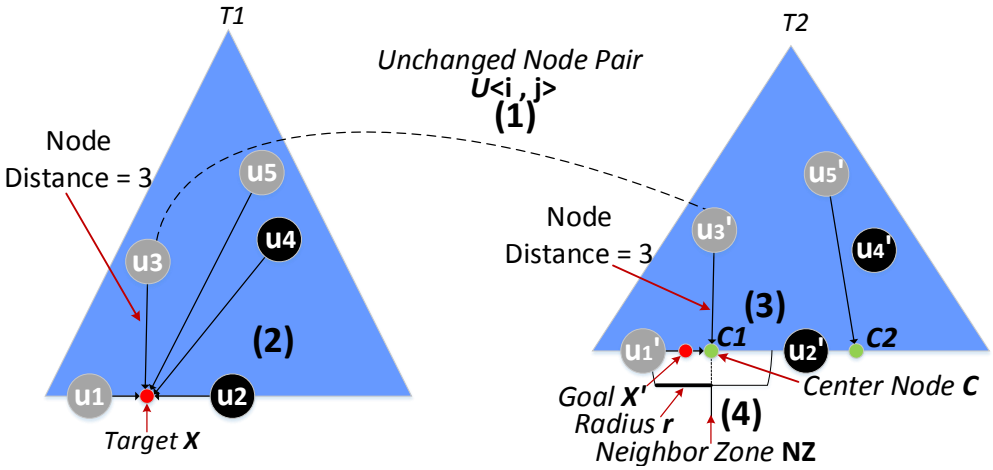


Figure 4.2: **Overview of Acquiring the Neighbor Zone**

pair set U which is a set of node pairs between T_1 and T_2 satisfying U = {<i, j> | node $T_1[i]$ and node $T_2[j]$ are matched (defined in Section 4.3.1).}.

- There are some page elements in a web page, and the relative positions between two of them remain unchanged or slightly changed in page variants. We refer to measurements of the position between two nodes in the HTML tree as *node distance* which is a set that consists of distances' values from different dimensions.

As shown in Figure 4.2, let <i, j> be an unchanged node pair in U. We expressly write the unchanged node pair <i, j> as U<i, j> by adding a prefix U, in order to make it clear that <i, j> is an unchanged node pair. Similarly, we use U<i> and U<j> (half part of U<i, j>) to denote T_1<i> and T_2<j> respectively.

(1). We first compute the unchanged node pair set U from T_1 and T_2.

(2). For each U<i, j> in U, we calculate the node distance between U<i> and *Target* in T_1. Let the calculated result be d, i.e. node-distance (U<i>, *Target*) = d.

(3). Then we list up the leaf nodes from T_2 which have the same distance from U<j> and let the listed nodes be C_k (k=1,2,...), i.e. node-distance(U<j>, C_k) = d.

(4). At last, we select the optimal C_k as the *center node C* which is supposed to be the *Goal* or near the *Goal*. Leaf nodes on both sides of the center node (within a certain radius r) are the neighbor zone.

For instance, as shown in Figure 4.2, suppose that we have an unchanged node pair U<u_3, u'_3>. Let node distance between *Target X* and u_3 be 3, so 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 neighbor zone of C_1. The radius r is for the range of the neighbor zone.

In Sections 4.2, 4.3 we give details of each component.

4.2 Data Type and Node Distance of Page Elements

4.2.1 Data Type

Web pages display information through a variety of media such as text string, images, videos and etc. Such visible content lies in essence in the leaf nodes of the HTML tree, where a leaf node is a node with no children in the tree.

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 web page (except text in form elements or custom embedded objects) 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 the characteristic by three kinds of information: *property*, *affiliation* and *structure*.

- **Property** is *text*, *image*, *video*, *audio* and *others*.
  - *Text* is the character string in web pages such as an article.
  - *Image*, *video*, *audio* are instances of the image, video, audio multimedia file respectively.

- *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 node <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*.

  - *Single occurrence* denotes a leaf node and its ancestor nodes do not have similar sibling nodes.
  - *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 4.3, the *structure* is different in the *detail page* and the *list page*. Figure 4.3(a) is a detail page corresponding to the HTML tree in Figure 1.2 which focuses on presenting a single object. Figure 4.3(b) is a list page which contains similar objects in a sequential way.

4.2.2 Node Distance

The *node distance* depicts the layout relationship between a leaf node (e.g.,*Target*) and other nodes in an HTML tree. As illustrated in Figure 4.4, the layout of a web page is usually displayed in the left-to-right, top-to-bottom order. Correspondingly, the inner structure of the

(a) A Detail Page

(b) A List Page

Figure 4.3: **Same Data (product name) in Different Structures of Sample Pages** (a) Single Occurrence Structure. (b)Sequential Occurrence Structure.

page is a rooted ordered labeled HTML tree whose node order is significant, where leaf nodes carry visible contents of the page and intermediate nodes are used to control the internal logical structure or the display form of the content.

As illustrated in Figure 4.3(a) and Figure 1.2 (a web page and its HTML tree structure), the path layer of a page element starting from the root node indicates the “height” level of the

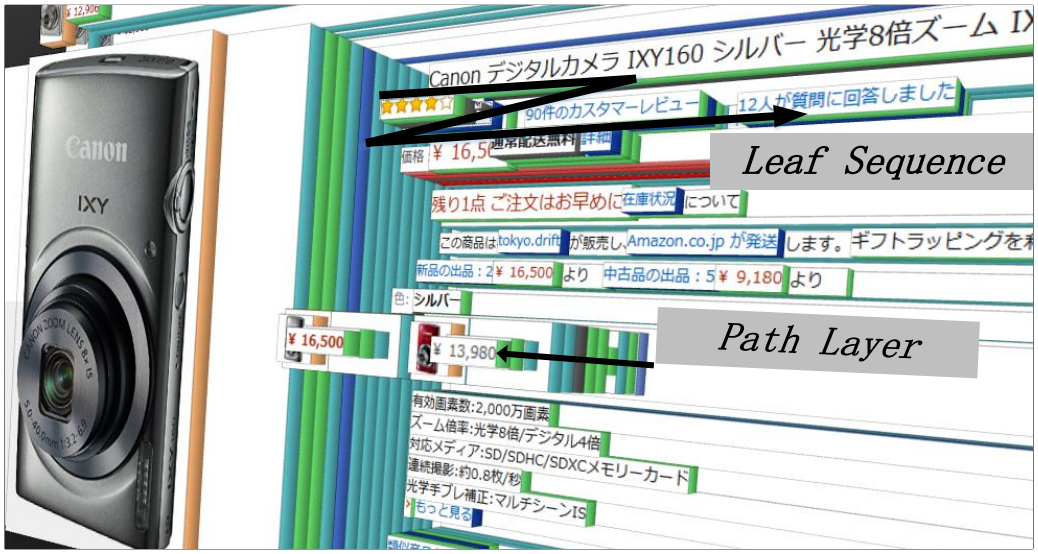


Figure 4.4: **Layout Relationship : Leaf Sequence and Path Layer**

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 ⑫ of HTML tree in Figure 1.2. 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 web page.
- The path layer of two nodes in the HTML tree controls the inner height distance of the nodes.

For example, the leaf sequence between node ⑤ and ⑨ is 5,6,7,8,9 and the path layers of both these two nodes are 8.

Based on these two features, we develop the node distance for nodes in HTML tree to represent their layout relationship. 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 values 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 $rml(n)$ 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 4.1 (Leaf Array / Relative Leaf Array) 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 $k \in [1, m]$ denotes the index value of the leaf node l_k in the leaf array. The relative leaf array is a sub-array of the leaf array whose elements have the same property as that of *Target*.

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 4.2 (Leaf Distance / Relative Leaf Distance) Let $idx(l)$ and $idx_{\text{relative}}(l)$ be index values of leaf node l in the leaf array and relative leaf array respectively. A leaf distance $leaf(l_a, l_b)$ between leaf l_a and leaf l_b is the number of leaf nodes between l_a(exclusive) and l_b(inclusive). It can be calculated by the following formula.

$$leaf(l_a, l_b) = idx(l_b) - idx(l_a) \tag{4.1}$$

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

$$leaf_{\text{relative}}(l_a, l_b) = idx_{\text{relative}}(l_b) - idx_{\text{relative}}(l_a) \tag{4.2}$$

For a node l_k whose property is different from the *Target*'s, toward the direction of the *Target*, e.g., rightwards, we search the first node l_{k+i} $(k + i \leq m)$ that has the same property as the *Target*'s. Then we specify the index of l_{k+i} in the relative leaf array as the relative index of the node l_k, i.e., $idx_{\text{relative}}(l_k) = idx_{\text{relative}}(l_{k+i})$. For example, a relative leaf array is illustrated in Figure 4.5, i.e., $[l_2, l_3, l_5, l_6, l_7]$, that have the same *property* as the *Target*'s, i.e., text. The property of node l_1 is img which is not the same as the text. Along the red arrow toward to *Target* in the figure, we will get node l_2 firstly whose property is the same as the *Target*'s. Therefore, $idx_{\text{relative}}(l_1) = idx_{\text{relative}}(l_2) = 1$.

Definition 4.3 (Leaf Boundary) For a leaf node l and a node n in tree T, the leaf boundary of the node 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.

$$LB(l, n) = \begin{cases} lml(n) & |leaf(l, lml(n))| \leq |leaf(l, rml(n))| \\ rml(n) & |leaf(l, lml(n))| > |leaf(l, rml(n))| \end{cases} \tag{4.3}$$

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

Definition 4.4 (Absolute Leaf Distance - *ALD*) The absolute leaf distance $ALD(l, n)$ in T is the leaf distance between l and n's leaf boundary $LB(l, n)$.

$$ALD(l, n) = leaf(l, LB(l, n)) \tag{4.4}$$

Definition 4.5 (Relative Leaf Distance - *RLD*) The relative leaf distance $RLD(l, n)$ in T is the relative distance between l and n's leaf boundary $LB(l, n)$, where the leaf nodes between l and $LB(l, n)$ must have the same property of data type with l.

$$RLD(l, n) = leaf_{\text{relative}}(l, LB(l, n)) \tag{4.5}$$

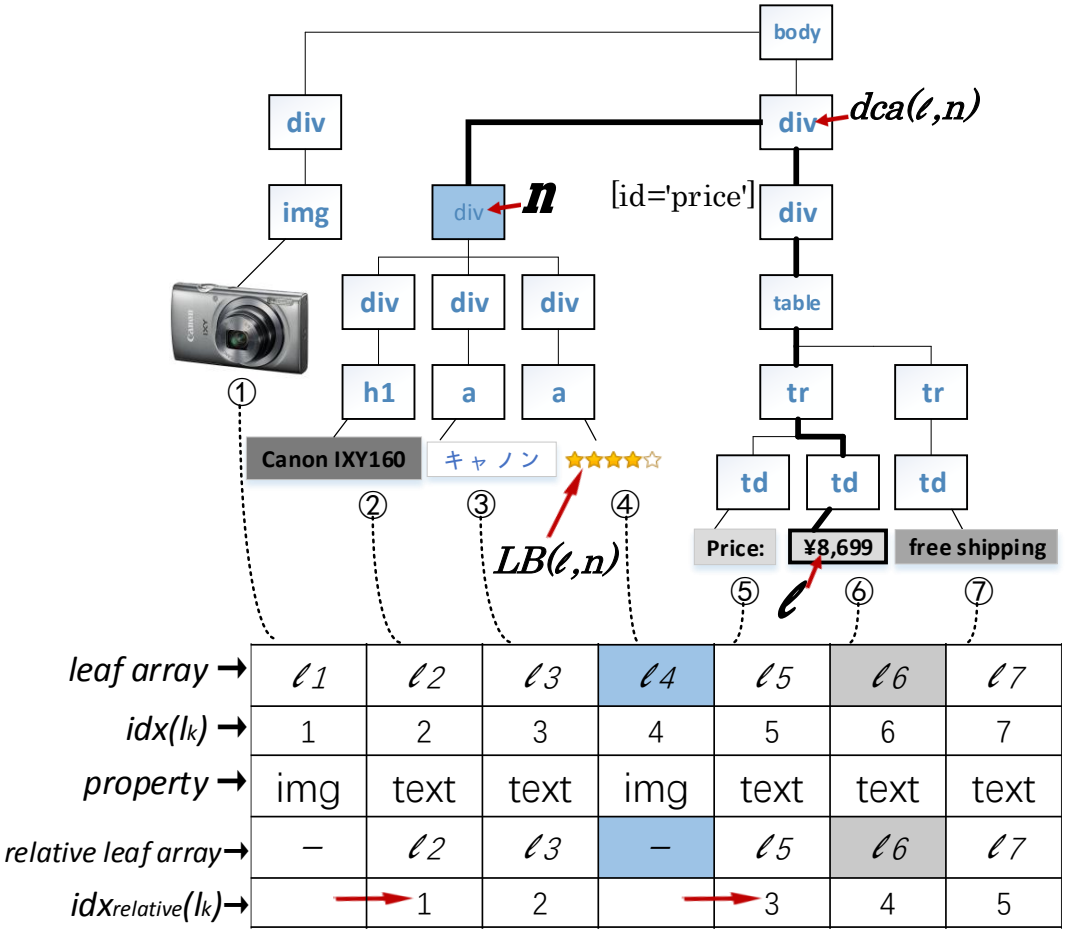


Figure 4.5: **An Example of Calculating Node Distance**

Node distances between a grey-colored leaf node l and a blue-colored intermediate node n in the subtree of Figure 1.2.

Definition 4.6 (Path Distance - *PD*) Let $dca(l, n)$ be the deepest common ancestor node, $edge(l, n)$ the number of edges between l and n. The path distance $PD(l, n)$ between l and n in T is the number of edges on the path from l to n via the node $dca(l, n)$.

$$PD(l, n) = edge(l, dca(l, n)) + edge(n, dca(l, n)) \tag{4.6}$$

For example, as shown in Figure 4.5, the leaf array and relative leaf array are illustrated at the bottom, i.e., $leaf\,array[l_1, l_2, l_3, l_4, l_5, l_6, l_7]$ and $relative\ leaf\,array[l_2, l_3, l_5, l_6, l_7]$ respectively. The boundary nodes of l and n are $lml(l) = rml(l) = 6$, i.e.,{⑥} and $lml(n) = 2$, $rml(n) = 4$, i.e.,{②, ④} respectively. Then for l, the leaf boundary of n, denoted by $LB(l, n)$, is the leaf node $rml(n)$, i.e.,④, because $|leaf(l,\ rml(n))| = |4 - 6| = 2$ which is smaller than $|leaf(l,\ lml(n))| = |2 - 6| = 4$. Therefore, we get $ALD(l, n) = leaf(l, LB(l, n)) = idx(LB(l, n)) - idx(l) = idx(④) - idx(⑥) = 4 - 6 = -2$. We get $RLD(l, n) = leaf_{\text{relative}}(l, LB(l, n)) = leaf_{\text{relative}}$ (⑥, ④)$= 3 - 4 = -1$. For $PD(l, n)$, we first get node $dca(l, n) = dca$(⑥, 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 $PD(l, n) = edge(l, dca(l, n)) + edge(n, dca(l, n)) = 5 + 1 = 6$.

4.3 Neighbor Zone of Designated Information

The neighbor zone represents a probable location of the node *Goal* in T_2 at the leaf node level. The definition of the neighbor zone is given as follows.

Definition 4.7 (Neighbor Zone) A neighbor zone is an ordered list of leaf nodes that are symmetrically distributed on both sides of a leaf node C namely *center node* (index c) within a radius r, i.e.,

$$NZ_{(c,r)} = leaf\,array[l_{c-r}, ...l_c, ...l_{c+r}]. \tag{4.7}$$

We use the following pseudo-code with Figure 4.6 together to depict the general process of neighbor 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., node-distance(l,n) = <$ALD(l, n)$, $RLD(l, n)$, $PD(l, n)$>, which are shortly denoted as <ALD, RLD, PD> 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 neighbor zone *NZ* that we defined in Eq 4.7.

Step 1 In line 2, We first compute the unchanged node pair set U by matching tree T_1 with T_2.

Algorithm 1: General Process of Neighbor Zone Acquisition

input : T_1; T_2; *Target* in T_1

output : Neighbor zone *NZ* in T_2

```
1 begin
2     U ← D(T1,T2);
3     Ck ← null (1 ≤ k ≤ m2) ;
4     for U<i, j> ∈ U do
5         d ← node-distance(U<i>, Target);
6         if node-distance(U<j>, Cjk) == d then
7             add all Cjk to Ck
8     C ← select most probable one from Ck;
9     return NZ(c,r) ← leaf array[lc−r,...lc,...lc+r]
```

Then we initialise a set of center 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.

Step 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*).

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

Step 4 Line 8 selects the most probable one (defined in Section 4.3.3) from C_k (k=1,...,m_2) as the center node C. Then, in line 9, let the nodes on two sides of C within the radius r be the neighbor zone *NZ*. Here, if $c - r < 0$, then let $c - r$ be 0; if $c + r > m_2$, then let $c + r$ be m_2.

For the illustration shown in Figure 4.6, 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

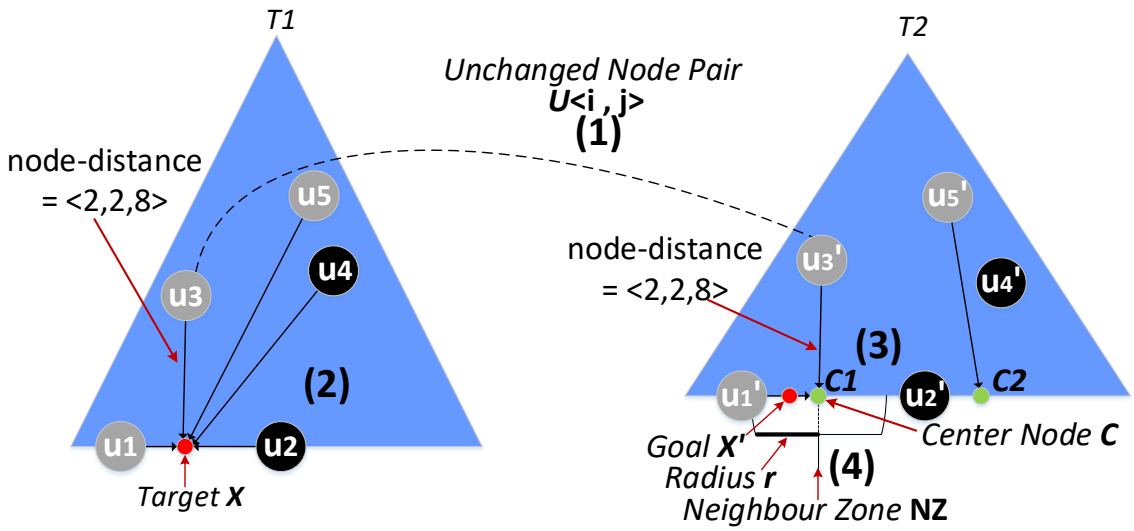


Figure 4.6: Illustration of Acquiring Neighbur Zone

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 neighbor zone of C_1.

We present the definition and calculation of unchanged node pair in Section 4.3.1 explaining Step 1; the computation of center node candidates in Section 4.3.2 covering Step 2 and Step 3; the selection of the final center node to generate neighbor zone in Section 4.3.3 addressing Step 4.

4.3.1 Unchanged Node Pairs

We compute the unchanged node pairs 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[88]. Such edit script will generate a minimum cost tree mapping which is defined as follows.

Definition 4.8 (Mapping) 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. For instance, as trees T_1, T_2 in Figure 1.2, the mapping prevents correspondence with multiple different nodes such as two node pairs {<①,①'>, <①,②'>}, etc. The two node pairs {<⑥,⑥'>} and {<⑦,⑦'>} satisfy the mapping definition, because ⑥ is the left sibling of ⑦ in T_1, and the ⑥' is also the left sibling of ⑦' in T_2. In other words, we cannot get two node pairs <⑥,⑦'> and <⑦,⑥'> in the mapping. The mapping between two trees can be generated by the tree edit distance algorithm as follows.

Definition 4.9 (Tree Edit Distance) 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 γ be a cost function on operations that γ(deletion) = c_1, γ(insertion) = c_2, γ(change) = c_3. The tree edit distance D(T1, T2) is the minimum cost of edit operations needed to transform T_1 to T_2 calculated as follows.

$$D(T_1, T_2) = c_1 \cdot n_d + c_2 \cdot n_i + c_3 \cdot n_c \tag{4.8}$$

The γ 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.

Then 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. The detailed procedure is presented as follows.

Step 1 We use TED algorithm to generate the minimum edit scripts (multiple) that transform T_1 to T_2. We select one script as E that generates mapping M.

Step 2 We find the node set $I\{i_1, i_2,..., i_n\}$ in T_1 which was not edited by the E.

Step 3 For each i_k (1≤k≤n), we find node set $J\{j_{k1},...,j_{km}\}$ in T_2 that satisfies $T_1[i_k] = T_2[j_{ku}]$ (1≤u≤m). From node set J, we select one node j_{ku} where $<i_k,j_{ku}>$ is in M.

Here we note that TED stores the nodes satisfying the conditions on the mapping during computing step 2. Therefore, we can use this mapping information to get the unchanged node pairs defined as follows without executing step 3.

Definition 4.10 (Unchanged Node Pair) The unchanged node pair U is a subset of tree mapping M between T_1 and T_2 satisfying $U = \{<i, j> \mid T_1[i] = T_2[j]\}$.

We implemented the dynamic programming computing process of tracing the unchanged node pairs of the zhang-shasha TED algorithm[88]. More information on the TED algorithm can be found in the original paper[88].

4.3.2 Center Node Generation

As the observation mentioned in Section 4.1, there are some page elements in a page that the relative position (leaf sequence and path layer) between two of them are unchanged or changed slightly in its page variants. Therefore, we use the node-distance (Section 4.2.2) and unchanged node pair (Section 4.3.1) to search possible locations of *Goal* on the tree structure. Such locations are represented by the center node candidate whose definition is given as follows.

Definition 4.11 (Center Node Candidate) Let <u_i, u_j> be an unchanged node pair between T_1 and T_2. For *Target* X of T_1, a center node candidate is a node C_k (k=1,2,...) of T_2, which satisfies $\{C_k \mid C_k$ is a leaf node and node-distance(u_j, C_k) = node-distance(u_i, X)$\}$.

4.3.2.1 An Example of Computing Center Node

We still use T_1 and T_2 in Figure 1.2 as an example. For simplicity, as shown in Table 4.1, we select nodes <④, ④'>, <⑥, ⑥'>, <⑧, ⑧'> and <div_price, div_price'> (abbreviation of node) to be the unchanged node pair set, which are illustrated in U<i>, U<j> columns respectively. For *Target* ⑥, we first compute the node-distance, i.e., <*ALD*, *RLD*, *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 ④', there is only one node, i.e.,⑤' that having the same *ALD* distance, i.e., -2. Likewise, the node with the same *RLD* distance is ⑥'. Therefore, for node ④', there exists none node in T_2 having the same node-distance triple. As a result, we get *center node* candidates C_1 = ⑤', C_2 = ⑥'.

4.3.2.2 Center Node Generation Algorithm

We present the neighbor zone center acquisition algorithm by the following pseudocode. Generally speaking, we apply node distance relationship, getting from *Target* and unchanged nodes in T_1, to unchanged nodes in T_2 to obtain *center node* candidates. which are pointed by the *stable nodes*.

The input are the *unchanged node pairs* $\mathbb{U}$ and the specified target node X. The output is a set of nodes which are *center node* candidates $\mathbb{C}$. We will get the *center node* ⑥ (the *Target*) in this example.

We first initialize the set of *center nodes* in line 2. Line (3-17) is the core process of the algorithm in which we traverse each *unchanged node pair* $U[i, j]$. Line(4-5) show if $U[i]$ is on the path from root of T_1 to X, e.g.,div_price, which means the target node X is a descendant

Table 4.1: **An Example of Center Node Generation (*Target* = ⑥)**

U<i>	node-distance	U<j>	nodes having the same distance	C_k
④	ALD = -2	④'	⑤'	none
	RLD = -1		⑥'	
	PD = 9		*inserted*, ⑤', ⑥', ⑦', ⑨', ⑩', ⑪', ⑫'	
⑥	ALD = 0	⑥'	⑥'	⑥'
	RLD = 0		⑥'	
	PD = 0		⑥'	
⑧	ALD = 2	⑧'	⑥'	⑥'
	RLD = 2		⑥'	
	PD = 8		*inserted*, ⑤', ⑥', ⑦'	
div_price	ALD = -1	div_price'	⑤'	⑤'
	RLD = -1		⑤'	
	PD = 4		*inserted*, ⑤', ⑥', ⑦', ⑧'	

node of $U[i]$, then we choose the deepest ancestor node to replace the *unchanged node pair* $U[i, j]$, i.e.,⑥, because whose *node distance* to X is the smallest.

In line (6-8) we first compute *node distance* tuple $\{ALD, RLD, PD\}$ for the node $U[i]$ in T_1. For example, for *Target* ⑥, the distance tuple of nodes ④, ⑥, ⑧ are {-2,-1,9}, {0,0,0}, {2,2,8} respectively. Then we apply the distance to the corresponding *unchanged node* $U[j]$ in T_2 where $LB'(U[j])$ denotes the direction is the same with the situation of $U[i]$ and X in T_1. For example, if the *leaf boundary* $LB(X, \text{div\_price}) = LB(⑥, \text{div\_price}) = lml(\text{div\_price}) =$ ⑤ in T_1, then $LB'(U[j]) = lml(U[j]) = lml(\text{div\_price}) = inserted$ in T_2 (not the $rml(U[j])$ = ⑦). The $Y_{\text{Ald}}(i, j)$, $Y_{\text{Rld}}(i, j)$ and $\mathbb{Y}_{\text{Pd}}(i, j)$ are the resulting leaf nodes of applying ALD RLD and PD on $U[j]$ respectively, where Y_{Ald} and Y_{Rld} are two nodes uniquely determined and $\mathbb{Y}_{\text{Pd}}$ is

Algorithm 2: Neighbor Zone Center Acquisition

input : unchanged node pairs $\mathbb{U} \leftarrow D(T_1, T_2)$, target node X;

output : center node candidates with meta data $\mathbb{C}$;

```
1  begin
2      ℂ ← null ;
3      for U[i, j] ∈ 𝕌 do
4          if U[i] in Path(root, X) then
5              U[i, j] ← getDeepestUnchangedNode(X, 𝕌, T1)
6          Y_Ald(i, j) ← ALD(X, U[i]) + LB′(U[j]);
7          Y_Rld(i, j) ← RLD(X, U[i]) + LB′(U[j]);
8          𝕐_Pd(i, j) ← PD(U[i], X) + LB′(U[j]);
9          if Y_Ald ∈ 𝕐_Pd then
10             if Y_Ald = Y_Rld then
11                 Y_IJ ← Y_Ald(i, j);
12                 SN_strong[i, j] ← U[i, j];
13                 add (Y_IJ, count++, SN_strong[i, j]) to ℂ_ARP;
14         else if Y_Rld ∈ 𝕐_Pd then
15             Y_IJ ← Y_Rld(i, j);
16             SN_weak[i, j] ← U[i, j];
17             add (Y_IJ, count++, SN_weak[i, j]) to ℂ_RP;
18     if 𝔸_ARP not empty then return ℂ ← ℂ_ARP ;
19     else if 𝔸_RP not empty then return ℂ ← ℂ_RP ;
20     else return ℂ ← 𝕐_Ald ;
```

a set of leaf nodes because we may get multiple nodes that have the same path distance from different anchor nodes in the XPath of node $U[j]$. For example, the resulting nodes tuple of applying node distance tuple on each $U[j]$ of ④, ⑥, ⑧ are {⑤, ⑥, {*inserted*, ⑤, ⑥, ⑦, ⑨,

⑩, ⑪, ⑫}}, {⑥, ⑥, ⑥}, {⑥, ⑥, {*inserted*, ⑤, ⑥, ⑦}} respectively.

Line(9-13) tell because all distances on $U[i]$ in T_1 point to node X, then if all distances on $U[j]$ in T_2 also point to the same node Y_{IJ}, the $U[i, j]$ is called as a *strong stable node* denoted by $SN_{\text{strong}}[i, j]$, e.g.⑥, ⑧ in T_1, T_2 respectively, and the node Y_{IJ} is a *center node* candidate, i.e.,⑥ in T_2. The *count* denotes how many *stable nodes* point to Y_{IJ}, e.g., *count* = 2 where Y_{IJ} = ⑥ that is pointed twice by nodes ⑥ and ⑧.

Line(14-17) present *weak stable nodes* pointing to *center node* Y_{IJ} which is a backup in case we can not get *strong stable nodes* when page change dramatically.

Line (18-20) mean if the *strong stable nodes* set is not empty then we return the *center node* candidate set $\mathbb{Y}_{IJ}$ with the count value and so on.

4.3.3 Neighbor Zone Acquisition

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

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

Therefore, the question is how to select the most probable one as the center node C. Intuitively speaking, we choose the that is closest to the desired node or is generated the most times. For instance, as shown in Table 4.1, U<div_price, div_price'> and U<⑥, ⑥'> generated C_1 = ⑤', C_2 = ⑥' respectively. Here the C_2 are generated by U<⑥, ⑥'> and U<⑧, ⑧'> twice. For such case that a center node candidate generated by multiple unchanged node pairs, we consider the average distance. Let *count_k* be the number of how many times of a center node candidate C_k (k = 1,2,...) generated by unchanged node pairs <u_1, u_1'>,...,<u_i, u_i'> (i ∈ [1, *count_k*]), then we select the center node C by the following two conditions.

- *Near the Target*. We first select the one that has the minimum value of the average *ALD*

between the *Target X* and the corresponding U<i> in T_1, denoted as follows.

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

- *Majority Voting*. If still exist multiple C_k that have the same minimum value of the average *ALD*, we select the one that has the max value of the $count\_k$, denoted as follows.

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

For the example shown in Table 4.1, 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., $|-1|/count\_1 = 1$, $|0+2|/count\_2 = 1$. Then because count_2 is larger than count_1, we choose the C_2, i.e., ⑥' as the center node C.

Chapter 5

Path Similarity based Extraction Method

In this chapter, we present the stable extraction approach based on the *path similarity*. The method was inspired by an observation as follows.

- The element characteristic of the *Goal* is more similar to the *Target* than other nodes of page variants.

In other words, as shown in Figure 1.3, the X' of T_1 is more similar to the X than other nodes of T_2. Therefore, we can search the *Goal* in the leaf nodes of the page vairant by ranking the characteristic similarity.

The rest of this chapter is organized as follows. An overview of the approach is given in Section 5.1. The detailed similarity measurements of page elements are addressed in Section 5.2, 5.3, 5.4, 5.5.

5.1 Overview of Path Similarity based Extraction Method

As shown in Figure 5.1, we focus on searching the *Goal* from the leaf nodes of page variants. We measure the similarity between the *Target* and specified leaf nodes of the page variant based on the characteristic information that derived from their XPath (Compute Path Similarity in Figure 1.4).

Firstly, we filter out nodes of the leaf nodes whose *property* of *data type* is different from the *Target*'s. Then we compute the similarity 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 the *Goal*. If more than one candidate corresponds to the highest score, we select them together.

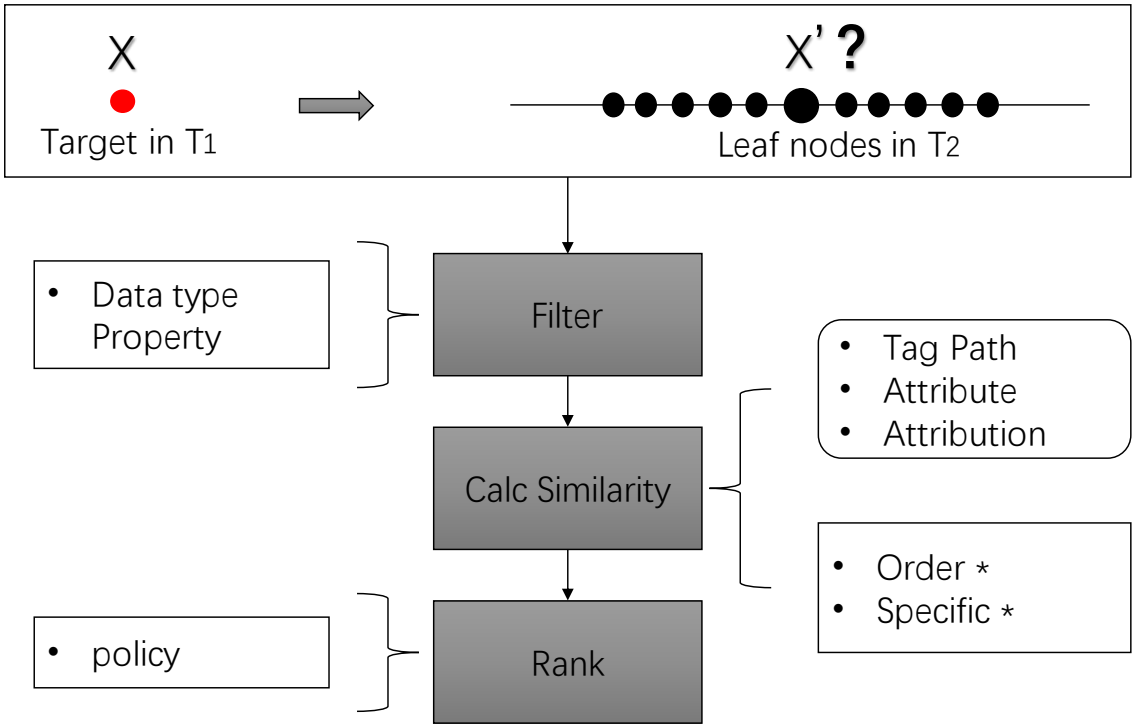


Figure 5.1: **Overview of the Path Similarity Method**

We use the following format to describe an XPath information: $N_1[O_1][@A_{1-0} = "V_{1-0}"][...][@A_{1-m_1} = "V_{1-m_1}"]/.../N_n[O_n][@A_{n-0} = "V_{n-0}"][...][@A_{n-m_n} = "V_{n-m_n}"]/N_{n+1}...$, 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_n} is the name of the m_{th} attribute of the n_{th} node; V_{n-m_n} is the corresponding value of attribute A_{n-m_n}; and N_n is the parent node of N_{n+1}. Because O_n

and A_{n-m_n} 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 in the following subsections. For emphasizing the differentiation between paths, we omit the head part which every absolute XPath contains, i.e., html and body tags and their attributes.

5.2 Tag Path Similarity

Following our XPath definition, the tag path is defined as $N_1[O_1]/\ldots/N_n[O_n]/\ldots$ which uniquely describes the position of a node in an HTML tree. For instance, the tag path of leaf node ⑥ in Figure 1.2 can be denoted as (omit html, body and the same below):

- P_a = /div[2]/div[2]/table/tr[1]/td[2]/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 other paths 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 $N_1/\ldots/N_n/\ldots$ to distinguish the tag sequence. For example, as shown in Figure 1.2, leaf nodes ⑤~⑦ and ⑨~⑫ share the same tag path cut $P_$cut shown below.

- $P_a_$cut = /div/div/table/tr/td/text()

5.2.1 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. Given two strings s_a and s_b on an alphabet Σ (e.g. the set of ASCII characters), the string edit distance $\delta(s_a, s_b)$ is the minimum number of basic edit operations that transforms s_a into s_b. In the SED algorighm, the unit used to

be compared between two strings is one single character in the string on the alphabet. For example, in string s = "abcd", the units in the string for SED computing are 'a', 'b', 'c', 'd' four characters. Since it can be seen as a special case of the TED algorithm which was introduced in Section 3.3, 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 [89].

We treat the content of each layer within the tag path as a character in the string to get the tag path edit distance score $TPED(P_a, P_b)$ between tag paths P_a and P_b. The content of each layer of the path is on the HTML tag name set (e.g. div, img, a, etc.). The tag path edit distance $\delta(P_a, Pb)$ is the minimum number of basic edit operations (deletion, insertion, change) that transforms P_a into P_b. The detail of HTML tag name set can be found in the HTML Element Reference of W3C `https://www.w3schools.com/tags/`. Here, the unit of TPED is the tag name and the sibling order number (if exist) in one path layer in the tag path sequence. For example, the units of P_a is div2,div2, table, tr1, td1, text(). Then the $TPED(P_a, P_b)$ is computed as follows.

$$TPED(P_a, P_b) = 1 - \frac{\delta(P_a, P_b)}{Max(|P_a|, |P_b|)} \tag{5.1}$$

TPED score reaches its best value at 1 (identical paths) and worst at 0.

where $|P_a|$, $|P_b|$ denote the length (i.e.number of path layer) of P_a, P_b respectively.

For example, in Figure 1.2, 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[1] to td[2]) and $\delta(P_a\_cut, P_b\_cut) = 0$. All lengths of P_a, P_b and the corresponding tag path cut are 6. Therefore, we get $TPED(P_a, P_b) = 1 - \frac{1}{6} = 0.833$, $TPED(P_a\text{\_cut}, P_b\text{\_cut}) = 1 - \frac{0}{6} = 1$.

5.2.2 Common Tag Path Distance Score (CTPD)

Intuitively, the TPED focuses on measuring how similar between two tag sequences. The common tag path distance (CTPD) calculates the ratio of how much the continuous common part between two paths from the beginning of paths. In addition, the CTPD also concerns the tag difference in the higher path layer (close to the leaf node) will have a greater impact (more distinctive and exclusive) on the score than the difference happened in the lower path layer.

As illustrated in Figure 5.2, the common path CP between two tag paths P_a, P_b is the identical tag sequence that start from the head of paths continuously. We denote them as follows.

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

Then the common tag path distance score ($CTPD$) of tag path P_a and P_b is defined as follows.

$$CTPD(P_a, P_b) = \frac{|CP(P_a, P_b)|}{|CP(P_a, P_b)| + |Diff_a| + |Diff_b|} \tag{5.2}$$

The CTPD is a normalization value that reaches its best value at 1 and worst at 0 (no common tag path) and is straight to be understanded. By the derivative equation, we can see that the CTPD satisfy the "the higher layer node has more impact on the score" requirement. We still use node ⑥ (P_a) and node ⑤ (P_b) in Figure 1.2 as an example, whose tag path and tag path cut are listed in Section 5.2.1. 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 $CP(P_a\_\text{cut}, P_b\_\text{cut})$ = /div/div/table/tr/td/text() whose corresponding $Diff_a$ and $Diff_b$ are empty (length = 0). Therefore, $CTPD(P_a, P_b) = \frac{4}{4+2+2} = 0.5$, $CTPD(P_a\_\text{cut}, P_b\_\text{cut}) = 1$.

In summary, we compute the final tag path similarity score, denoted by $TAG(P_a, P_b)$, of two tag paths as following formula. For brevity, let $TPED$ represent $TPED(P_a, P_b)$, $TPED\_\text{cut}$

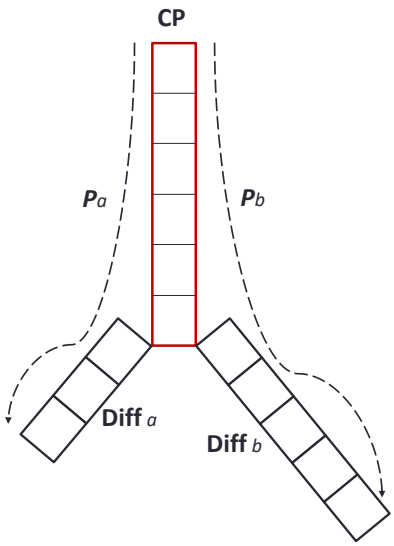


Figure 5.2: **Common Tag Path Distance**

The $CP(P_a, P_b)$ is the common path sequence between P_a and P_b. The $Diff_a$, $Diff_b$ are the remaining subpath that P_a, P_b remove CP respectively.

represent $TPED(P_a\_\text{cut}, P_b\_\text{cut})$, as well as $CTPD$ and $CTPD\_\text{cut}$. Let e_1, e_2, e_3, e_4 be weight coefficients of these four tag path scores respectively.

- If the structure is single

$$TAG(P_a, P_b) = \frac{e_1 \cdot TPED + e_2 \cdot TPED\_\text{cut} + e_3 \cdot CTPD + e_4 \cdot CTPD\_\text{cut}}{\sum_{i=1}^{4} e_i} \tag{5.3}$$

- If the structure is sequential

$$TAG(P_a, P_b) = \frac{e_2 \cdot TPED\_\text{cut} + e_4 \cdot CTPD\_\text{cut}}{\sum_{i=1}^{2} e_i} \tag{5.4}$$

As we mentioned abovce, the TPED and CTPD measure the similarity between two paths from two separate dimensions. Here, we think that the TPED (including cut and non-cut) and CTPD (including cut and non-cut), i.e., the two dimensions, are equally important, therefore,

for the sake of generality, the weights are set to 1 equally. However, the weight parameters are added here for future extension studies. We will record and analyse the influence of different weight parameters on the results in large-scale extraction tasks. Following the example of ⑤ (P_a) and ⑥ (P_b) in Figure 1.2 (the structure is single), the $TAG(P_a, P_b) = \frac{0.833+1+0.5+1}{4} = 0.833$.

5.3 Attribute Path Similarity

Each element of an XPath can have attributes which are used to define some 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 by "=". Following the definition of XPath, the attribute path is addressed as $[@A_{1-0} = "V_{1-0}"][...][@A_{1-m_1} = "V_{1-m_1}"]/.../[@A_{n-0} = "V_{n-0}"][..][@A_{n-m_n} = "V_{n-m_n}"]/[@A_{(n+1)-0} = "V_{(n+1)-0}"]...$

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 1.2 is as follows, where the "__" denotes the attribute is empty.

- AP_a = /id="container"/id="price"/id="main" 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, as shown in Figure 5.3, 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 Eq 5.5 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

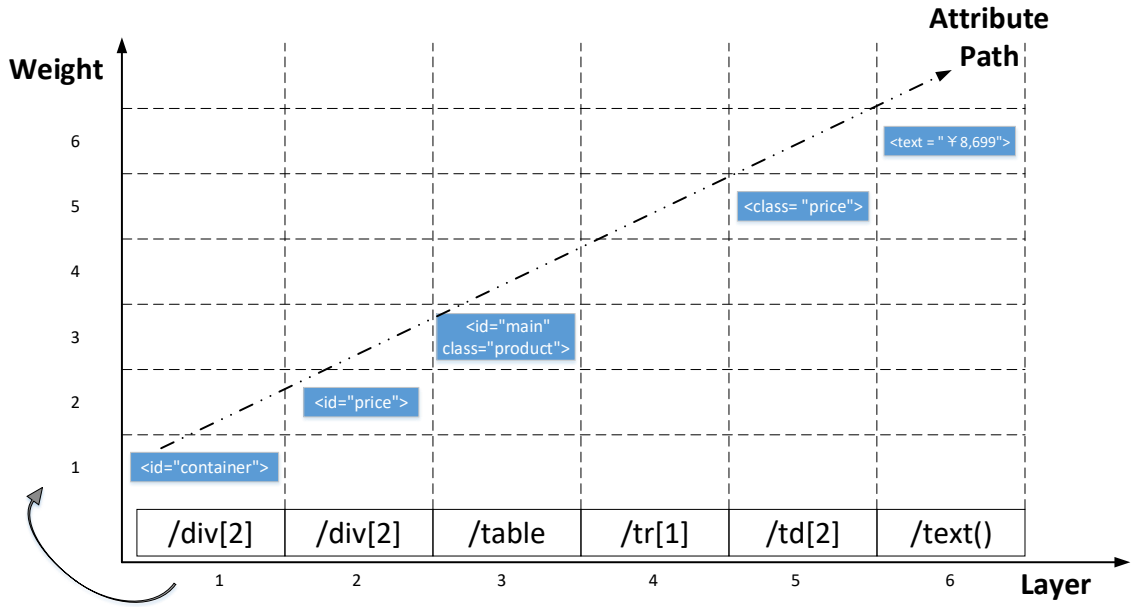


Figure 5.3: **Layer and Weight in Attribute Path**

attribute path can be matched with that of other attribute paths.

$$ATTR(AP_a, AP_b) = \frac{\sum_{i=1}^{n} \sum_{j=1}^{m_i} \frac{W_i}{m_i} unitAttrScore(AP_a, i, j, AP_b)}{\sum_{i=1}^{n} W_i} \tag{5.5}$$

Let AP_a 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 5.4, we will get two attributes of the 3_{rd} layer, i.e., id="main" and 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 $attr(AP, i, j)$ be the j_{th} unit attribute of path layer i of the attribute path AP. For example, $attr(AP_a, 3, 2)$ = 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 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.

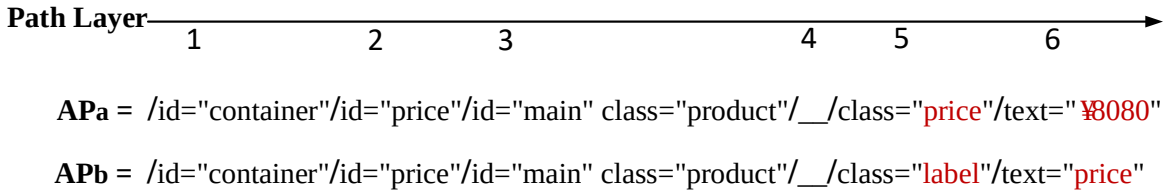


Figure 5.4: **An Example of Attribute Path with Path Layers**

In real-world web pages, 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 use semantic techniques like word segmentation, *Jaccard* coefficient (intersection over union), SED score, etc., to make the approximate match.

5.3.1 Attribute Path Similarity Algorithm

We present attribute path similarity algorithm in the following pseudocode.

The input are two attribute paths AP_1 and AP_2 where AP_1 is treated as a base path that is to be compared with another path, e.g., the attribute path of the Target. The output is the attribute similarity score which gets its best value at 1 and worst at 0. Line (2-3) parse attributes of two paths into unit attribute list with weight information with a function in Line (14-26). For example, an attribute path of node ⑥ in Figure 1.2 is illustrated in Figure 5.3. The path has 6 layers denoted by *layerNum*. The weight of each layer is equal with its layer value. Therefore, the *weightSum* of this path is 1 + 2 + ... + 6 = 21. The *subAttr* means all attributes in a layer can be splitted into unit attributes. For instance, the layer 3 in Figure 5.3, the *subAttr* is <id = "main" class = "product"> whose two unit attributes are id = "main" and class = "product", which

Algorithm 3: attribute path score

```
input  : Attribute path AP_1, AP_2
output : AP_score

1  begin
2      List unitAttrs1 ← getUnitAttributeListWithWeight(AP_1);
3      List unitAttrs2 ← getUnitAttributeListWithWeight(AP_2);
4      AP_score ← 0;
5      for u_1 ∈ unitAttrs1 do
6          subScore ← = u_1.getUnitWeight();
7          for u_2 ∈ unitAttrs2 do
8              unitScore ← calcUnitAttributeSim(u_1, u_2);
9              isMatch ← attrMatchPolicy((u_1, u_2, unitScore);
10             if isMatch is true then
11                 AP_score += subScore;
12                 break;
13     return AP_score;
14 Function getUnitAttributeListWithWeight(AP):
15     layerNum ← Layer(AP);
16     weightSum ← Σ_{i←1}^{layerNum} i;
17     for i ← 1 to layerNum do
18         subAttr ← AP[i];
19         currentLayer ← i;
20         unitAttrsOfCurrentLayer ← splitAttr(subAttr);
21         siblingAttrNum ← unitAttrsOfCurrentLayer.size();
22         for j ← 1to siblingAttrNum do
23             unitAttr ← unitAttrsOfCurrentLayer(j);
24             unitWeight = currentLayer/weightSum * 1/siblingAttrNum;
25             unitAttrsList.push(unitAttr, unitWeight)
26     return unitAttrsList;
27 //compute the unit attribute similarity score;
28 Function calcUnitAttributeSim (u_1, u_2);
29 //judge whether two unit attribute is similar based on the score;
30 Function attrMatchPolicy (u_1, u_2, score);
```

share the weight of the layer. Therefore, the weight of *unitAttr* id = "main" is computed by the formula in Line 24, i.e., $unitWeight = \frac{3}{21} * \frac{1}{2} = 0.071$.

Line 4 initialize the attribute score ATTR_{score}. Line(5-8) use a function,denoted by *calcUnitAttributeSim*, to calculate similarity score namely *unitScore* between each unit attribute of the base attribute path AP_1 (*Target*) and AP_2. Line(9-12) use a function, denoted by *attrMatchPolicy* to judge whether two unit attribute is similar based on their *unitScore*. And if they are matched, then the *unitWeight* of current unit attribute of AP_1 ,shown in Line 6, is added to the attribute score. Line 13 returns final result of attribute score.

An important component of this algorithm is the unit attribute similarity score function (*calcUnitAttributeSim*) whose general process is briefly introduced as follows.

- If attribute names *A* of two unit attributes are different, the unit attribute score is set to 0.
- If attribute name is id or class, first we segment the attribute values and then compute the *Jaccard* coefficient (intersection over union) as the unit attribute similarity score.
- If attribute name is href or src, we clean the url string first and then compute the SED value as the unit attribute similarity score.
- For other attribute names, we compute the SED value as the unit attribute 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. Furthermore, for some

content oriented site, such as web news page, the semantic similarity between attributes are also concerned.

5.3.2 An Example of Attribute Path Similarity

Figure 5.4 shows an example of calculating attribute path similarity. We use the attribute paths of node ⑥ (base AP_a) and node ⑤ (AP_b) as an example. The base attribute path AP_a has 6 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 $\sum_{i=1}^{6} W_i = 1+2+\ldots+6 = 21$. Obviously, except the 5_{th} and 6_{th} of AP_a whose attribute values are written in red, 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 calculated as follows. $\frac{\frac{1}{1}*1+\frac{2}{1}*1+\frac{3}{2}*1+\frac{3}{2}*1+\frac{4}{1}*1+0+0}{21} = \frac{10}{21} = 0.476$.

5.4 List Order Similarity

When the target node is located in the sequential structure, e.g., the product name of the first item in Figure 4.3(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 5.5 where the li[i] ($i = 1, 2, \ldots, 9$) 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 labeled *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

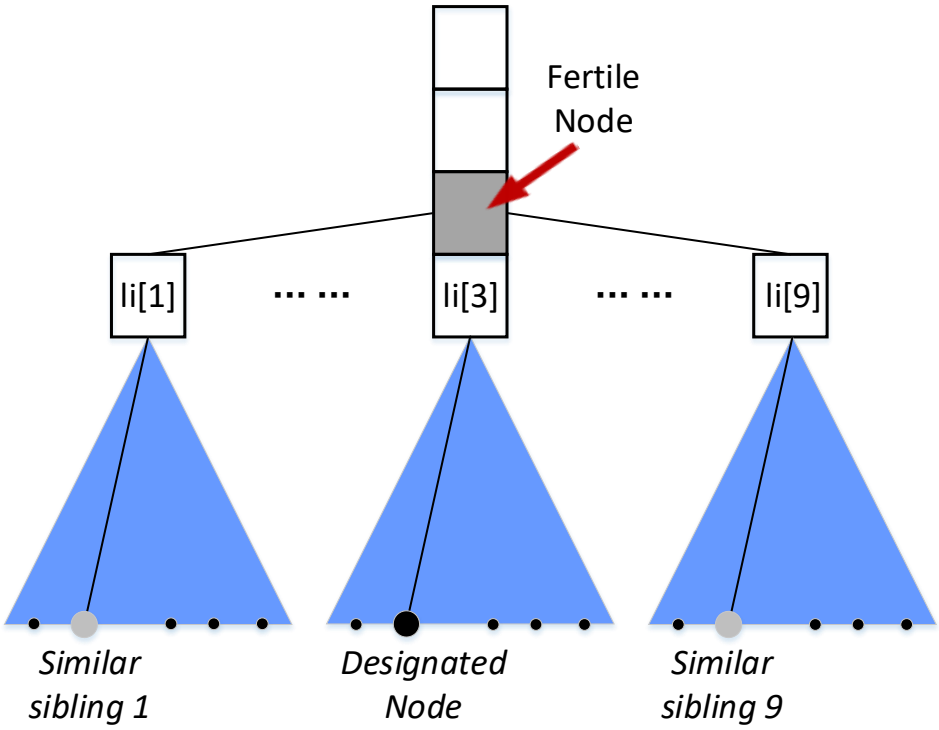


Figure 5.5: **List Order of a Designated Node in the Sequential Structure**

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 5.5, 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 $\frac{3}{9}$.

After obtaining the list order of two nodes, we compute the list order similarity score $ORDER(P_a, P_b)$ by Eq 5.6 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.

$$ORDER(P_a, P_b) = (1 - \frac{|SIB_a - SIB_b|}{max(NUM_a, NUM_b)}) \cdot (1 - \frac{|NUM_a - NUM_b|}{max(NUM_a, NUM_b)}) \tag{5.6}$$

For example, let the tag path of the designated node in Figure 5.5 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 into Figure 5.5. We select the 3_{rd} 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 $(1 - \frac{|3-3|}{max(9,10)}) \cdot (1 - \frac{|9-10|}{max(9,10)}) = 1 \cdot 0.9 = 0.9$.

As illustrated in Figure 5.5, the main process of calculating the list order similarity is as follows.

Step1 Fetch all ancestor nodes of the specified node(e.g., the *Target*).

Step2 Find the fertile nodes whose amount of child nodes is larger than a threshold th (e.g.,th = 5).

Step3 Fetch all descendant leaf nodes whose tag path passes through a fertile node.

Step4 Calculate the tag and attribute similarity between these descendants and specified node, and then rank them by similarity scores.

Step5 The nodes with the highest score are the brother nodes, which have very similar tag and attribute path structure with the specified node.

Step6 Get the list order of the specified node by $nodeSibling/nodeMaxSibling$.

For the specified node in Figure 5.5, we get *nodeSibling* = 3, *nodeMaxSibling* = 9 and therefore the list order is $3/9$.

5.5 Affiliation Similarity and Final Score

At last, we compute the affiliation score, denoted by $AFFI(P_a,P_b)$. As we presented in the Section 4.2, the affiliation of the data type is defined as the HTML element type of a leaf node

which is determined by ancestor nodes the leaf node affiliated to. It denotes that there exists specified ancestor node 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 an ancestor node <a>) is selected as the *Target*, then the corresponding *Goal* is probably still in a link environment in a page variant. Because such change usually is unlikely to happen, at first, we thought that once such change occurs, it should be given the greatest "punishment", for example, be directly filtered. However, in real extraction tasks, we found that when the structure of the page changes greatly (such as long time intervals), this kind of change still may exist. For example, the <table> used to control layout was changed to the . Correspondingly, we should not directly filter out the path just because their affiliation is different. Therefore, we let the greatest punishment be the score 0, otherwise, the award score be 1. Accordingly, the $AFFI(P_a,P_b)$ is calculated simply by judging whether tag paths P_a, P_b of two nodes contain the same affiliation value. If it contains, the score is set to 1, otherwise the score is set to 0.

Let XP_a, XP_b be the XPath of node a and b with the form defined in the head part of Chapter 5. The $PATHS(XP_a, XP_b)$ denote the path similarity score of XPath XP_a and XP_b, which 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 $ORDER(P_a, P_b)$. Let the Avg be a mean function, the final path similarity score is computed with the following formula.

$$PATHS(XP_a, XP_b) = Avg(TAG + ATTR + AFFI + ORDER) \tag{5.7}$$

We select the node(s) with the highest path similarity score as a result, i.e., the *Goal* node(s).

Chapter 6

Hybrid Extraction Mechanism and Implementation

In the previous Chapter 4 and Chapter 5, we presented two stable extraction methods. We will show their respective effectiveness in the next experimental chapter. While in the course of these experiments, we also observe that when dealing with page variants with large change, for example, the temperal change illustrated in Figure 6.1 (arrow marked with 2) where web pages changed with long time intervals, the SSN and SSP methods are not as effectively as they performed in the template changed page variants (arrow marked with 1). At the same time, we found that, for the largely changed (e.g.,arrow marked with 2 and 3) page variants, despite the effectiveness of directly locating the *Goal* node with the center node of the neighbor zone based method has been challenged, however, the *Goal* is still very probable in its neighbor zone. Therefore, we develop a hybrid extraction mechanism that combines the path similarity based method with the neighbor zone based method. In this situation, the neighbor zone denotes an approximate area of the *Goal* in page variants, which consists of a fragment of leaf nodes. And then we search the precise location of the *Goal* by the ranking the path similarity score of nodes within the neighbor zone.

In this chapter, we first present the change model of the page variant in Section 6.1, then we give an explanation of the hybrid extraction mechanism in Section 6.2, and at last we introduce

implementation details of a prototype of the hybrid method in Section 6.3.

6.1 Change Model of the Page Variant

As shown in Figure 6.1, there are three change directions, i.e.,arrow marked with 1, 2 and 3, that generate page variants which may lead to the stability problem. Suppose that the website has i templates for a page. For the template 1 at time t_1, it was specified a set of target data u_{11},u_{12},...,u_{1n}. The first change direction, i.e., template change, is that at time t_1, the user select and extract the target data (e.g.,. u_{11},...,u_{1n}) in template 1 and use the same XPaths to extract them (u_{21},...,u_{2n}) in template 2 which is similar but different with template 1. The second change direction, i.e., temporal change, denotes that at time t_1, the user select and extract the target data in template 1. And then at time t_2, the user directly reuse the same XPath to extract the desired data in the changed page denoted as template 1' consists of $u_{1'1}$,$u_{1'2}$,...,$u_{1'n}$. In the real extraction tasks, the more common change direction is "mixed" type, that is, a combination of the above two changing directions (grey arrow marked with 3). The page variants that result from this change direction, i.e., the future changed version of the template variant of the initial page, are even more challenging for the extraction.

6.2 Hybrid Extraction Mechanism

Corresponding to the change model shown in Figure 6.1, there are three kind of extraction tasks (scenario 1, 2 and 3). The three extraction tasks have different requirement on extraction methods. For the first kind of task 1, i.e., template page variants, even the website holds a huge number of pages, the amount of templates corresponding to the same topic are still limited and which usually have a very alike structure. Therefore the extraction mechanism should run quickly and have the ability to mitigate minor and medium structural variations of the page. For the second and the third kind of tasks 2, 3, i.e., temporal and mix changed page variants, the page's layout may change visibly in a future time point, accordingly a more stable mechanism

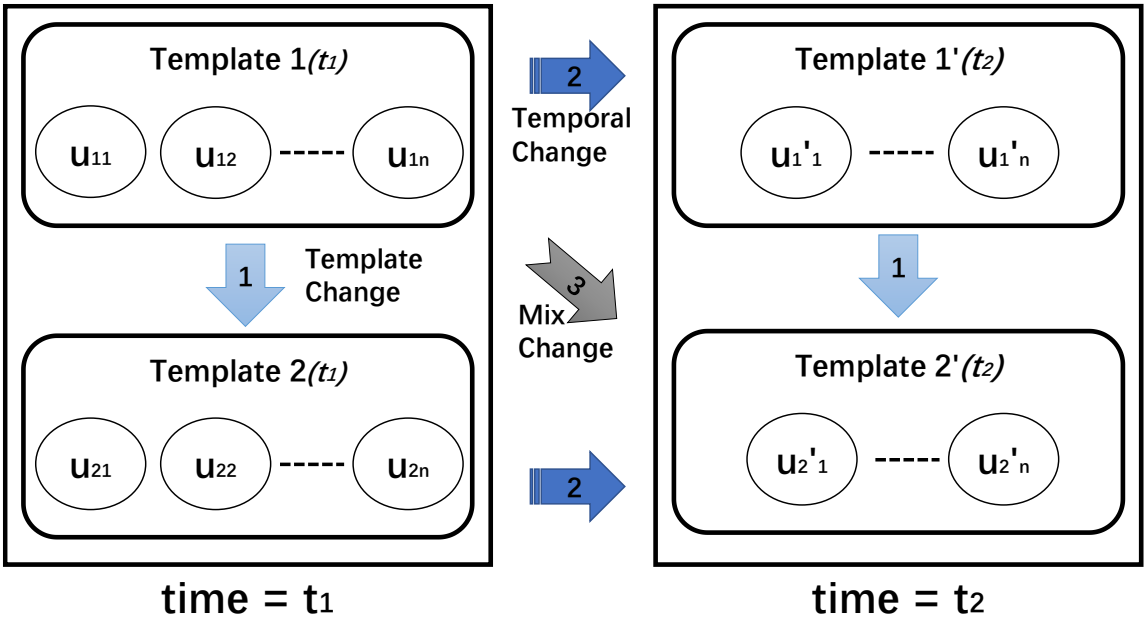


Figure 6.1: **Change Model of the Page Variant**

is needed to deal with such large structure changes.

Based on the explanation in the head of this chapter, a hybrid method is proposed as follows. As presented in the end of Section 4.3.3, we will get the center node C in the page variant. Therefore, as shown in Figure 4.2, we set a radius r to acquire all leaf nodes in the neighbor zone. The r can be determined by considering 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 10. We note that because we filter some nodes with their descendants in the HTML tree, the zone range may have some nodes that are not really leaf nodes. Therefore the real radius usually can be smaller than our empirical value. The architecture of the hybrid extraction method are shown in Figure 6.2.

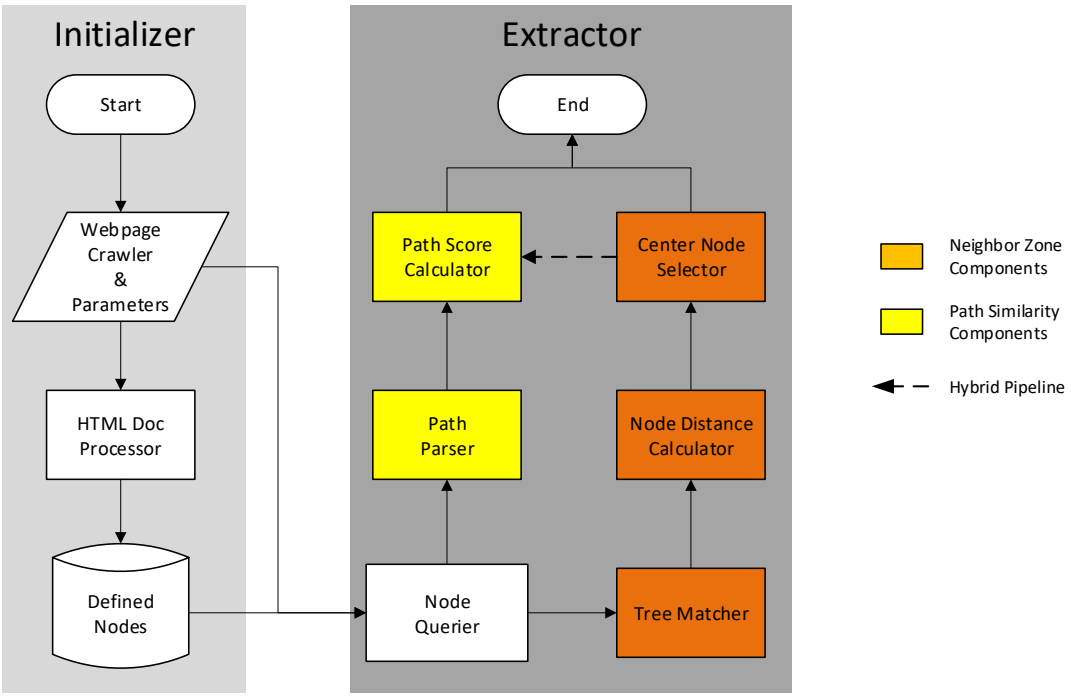


Figure 6.2: **Architecture of the Hybrid Extraction Mechanism**

6.3 Implementation

We implemented a prototype for the our methods. The inputs of the prototype are crawled HTML source files with designated target information, and two parameters : a property switcher and optionally a structure switcher. 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.

We use an HTML parser *Jsoup*[90](ver.1.7.2) The HTML Doc processor is used to parse and traverse each HTML page into a post-ordered labeled tree. Each node of the tree is recorded by a data structure named *Defined Node* and stored into a MySql database. As an example shown in Figure 6.3, the defined node record the following information of each node: pid,

tag, is_leaf, text, attributes, XPath_abs, path_attr, sibling_amount, src, lml_id, lml_tag, rml_id, rml_tag. The *pid* is the post-order number of the node in the ordered HTML tree. The tag is the tag name of the node. The is_leaf means whether the node is a leaf node. The XPath_abs records the absolute XPath of the node which can be used as a unique locator of it. The text records textual contents of the node. The attributes records textual contents of the node. The path_attr is the attribute path of the node. The sibling_amount is the number of sibling nodes. The src is the url value of the web page. The lml_id/rml_id and lml_tag/rml_tag are the pid and the tag of the left-most-leaf node/right-most-leaf node of the current node..

The node querier is an interface used to query nodes from the database. The neighbor zone and path similarity components (orange and yellow colored) are implemented as a pipeline style where each component works independently. The dotted arrow is the pipe that connects the neighbor zone component and path similarity component.

pid	is_leaf	tag	text		XPath_abs	attributes		s...	path_attr	
1	1	#comment	<!-- BEGIN COR...	35B	html/body/div[1]/c	comment=" BEGIN CORE...	40B	2	>+<>+<class="web-analy...	73B
2	1	#comment	<!-- END COREM...	25B	html/body/div[1]/c	comment=" END COREME...	30B	2	>+<>+<class="web-analy...	63B
3	0	div	<div class="we...	27B	html/body/div[1]	class="web-analytics...	24B	5	>+<>+<class="web-analy...	30B
4	1	img	<img src="/ima...	102B	html/body/div[2]/c	src="/images/HeaderF...	99B	1	>+<>+<id="mainbodydiv"...	199B
5	0	a	<a href="http:...	34B	html/body/div[2]/c	href="http://www.abe...	33B	1	>+<>+<id="mainbodydiv"...	97B
6	0	div	<div id="brand...	19B	html/body/div[2]/c	id="branding">_<	16B	4	>+<>+<id="mainbodydiv"...	61B
7	1	#comment	<!--signON/OFF-->	18B	html/body/div[2]/c	comment="signON/OFF">_<	23B	4	>+<>+<id="mainbodydiv"...	90B
8	1	#text	sign on	8B	html/body/div[2]/c	text=" sign o...	27B	1	>+<>+<id="mainbodydiv"...	165B
9	0	a	<a href="https...	67B	html/body/div[2]/c	href="https://www.ab...	68B	4	>+<>+<id="mainbodydiv"...	135B
10	1	#comment	<!--MMM-->	11B	html/body/div[2]/c	comment="MMM">_<	16B	4	>+<>+<id="mainbodydiv"...	83B
11	1	#text	my account	11B	html/body/div[2]/c	text=" my ...	33B	1	>+<>+<id="mainbodydiv"...	180B
12	0	a	<a href="http:...	76B	html/body/div[2]/c	href="http://www.abe...	77B	4	>+<>+<id="mainbodydiv"...	144B
13	1	#comment	<!--Basket-->	14B	html/body/div[2]/c	comment="Basket">_<	19B	4	>+<>+<id="mainbodydiv"...	86B
14	1	#text	basket	6B	html/body/div[2]/c	text="basket">_<	16B	1	>+<>+<id="mainbodydiv"...	157B
15	0	a	<a href="http:...	70B	html/body/div[2]/c	href="http://www.abe...	71B	4	>+<>+<id="mainbodydiv"...	138B

Figure 6.3: **An Example of Defined Node Structure (Fragment)**

Chapter 7

Experiment and Evaluation

In this chapter, we set up experiments on real-world web data extraction tasks to verify the following research questions shown in Section 7.1. The XPath based extraction[5, 6, 7, 8] is the baseline method which is compared with our approaches. Our goal is to design and implement extraction methods surpassing the XPath in terms of stability, which also can be flexibly used under different application scenarios.

The reason we focus on comparing with the XPath is as follows. The XPath is a powerful and the most widely used DOM-based HTML element locator. Almost most of other methods provided by DOM-based tools (e.g., CSS selector of Jsoup,) can be replaced by the XPath expression. In real-world extraction tasks, in many cases the XPath is the only choice that always can locates arbitrary web page element. Our methods are not aim to replace the XPath method, but to complement it and can be used in combination with it.

There is no "robust XPath" officially defined. Usually, a robust XPath expression is carefully evaluated by human professionals, which requires a lot of experience and efforts. We choose the XPath that can be automatically generated (e.g., by a plugin of the browser) as rivals. The relative XPath is considered more robust than the absolute XPath. In addition, we add the Robular(2014)[67] and Robular+(2016)[29] robust XPath generator as comparisons. As far as we know, these two methods are currently the most relevant work for automatic generation of

robust XPath in our application scenarios.

Many other path based methods are "ad hoc" style. We do not use these specific works because they are not easy to obtain for testing on the same datasets. To our understanding, we notice that each approach may have its own advantages which may not be replaced by others methods. Some comparison highly depends on the design purpose and the application environment used. Therefore, we classified the existing extraction methods in the related work (Section 1.3) and analyze the differences and limitations in our application scenario.

7.1 Research Questions

RQ1 Do the neighbor 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 center node of the neighbor 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 neighbor zone and path similarity components, can we get a more stable extraction method than using two components alone?

RQ2 If the page variant contains larger structural changes, can the neighbor zone, path similarity and the hybrid methods extract the desired information stably?

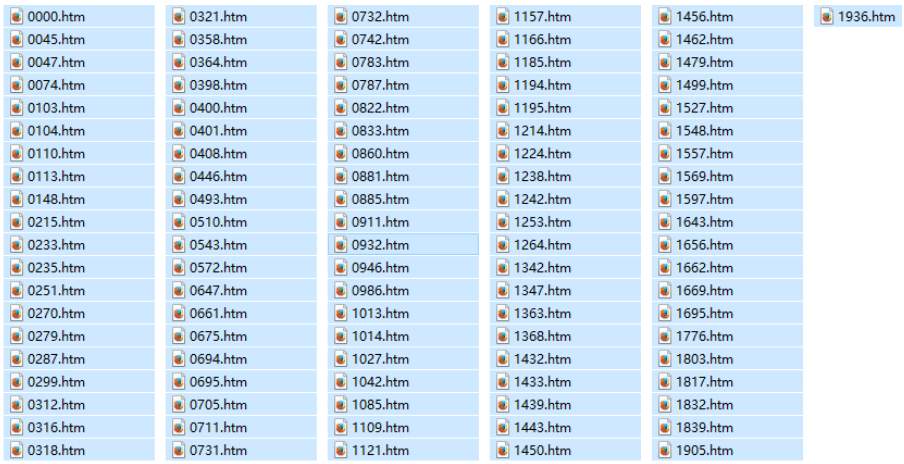
In order to measure the stability mentioned in RQ1 and RQ2, we need to observe whether the extraction methods have the ability to successfully extract the required data from page variants. We adopt the most commonly used evaluation criteria in this area, the F1-score (a weighted average of the precision and recall), as performance metrics, where 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.

7.2 Datasets

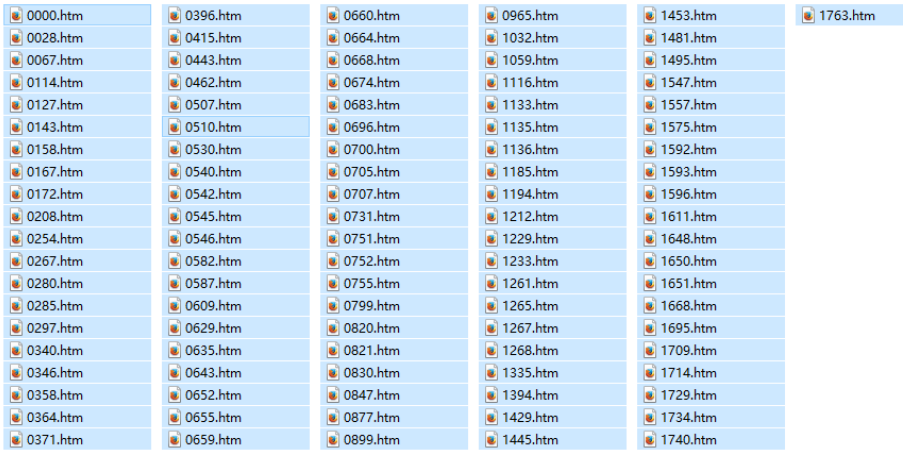
We prepared two datasets as a benchmark to compare the stability of our methods with the XPath based extraction. The first one, denoted by *Dataset1*, is gathered from a large dataset with ground-truth, publicly available at `http://swde.codeplex.com/`, published by Qiang Hao et al[21]. Because the source files of the dataset are very large (about to 8GB HTML that will generate 400GB MySQL defined nodes), we use a random sampling function (java.util.Random) to pick experimental sample pages. As summarized in Table 7.1, we randomly selected 101 pages (1 web page and 100 page variants) for each website to get totally 8,080 web pages from 80 diverse websites in 8 vertical fields such as autos, books, etc. Figure 7.1 shows an example of random picked samples from the *Autos* and *Cameras* verticals of the *Dataset1* that are used in the experiment.

Table 7.1: **The Composition of the *Dataset1***

Vertical	#Sites	#Pages	Target Data	websites
Autos	10	1,010	model, price, engine, fuel-economy	aol, autobytel, automotive, autoweb, carquotes, cars, kbb, motortrend, msn, yahoo
Books	10	1,010	title, author, ISBN-13, publisher, publish-date	abebooks, barnesandnoble, bookdepository, booksamillion, borders, christianbook,deepdiscount, waterstones
Cameras	10	1,010	model, price, manufacturer	amazon, beachaudio, buy, compsource, ecost, jr, newegg, onsale, pcnation, thenerds
Jobs	10	1,010	title, company, location, date	careerbuilder, dice, hotjobs, job, jobcircle, jobtarget, monster, nettemps, rightitjobs, techcentric
Movies	10	1,010	title, director, genre, rating	allmovie, amctv, boxofficemojo, hollywood, iheartmovies, imdb, metacritic,rottentomatoes
NBA Players	10	1,010	name, team, height, weigh	espn, fanhouse, foxsports, msnca, nba, si, slam, usatoday, wiki
Restaurants	10	1,010	name, address, phone, cuisine	fodors, frommers, gayot, opentable, pickarestaurant, restaurantica, tripadvisor,urbanspoon, usdiners, zagat
Universities	10	1,010	name, phone, website, type	collegeboard, collegenavigator, collegeprowler, collegetoolkit, ecampustours,embark, matchcollege, princetonreview, studentaid, usnews



(a) Auto-aol (101 of 2000)



(b) Camera-amazon (101 of 1767)

Figure 7.1: **An Example of Random Picked Samples from *Dataset1***

As shown in Table 7.2, 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*[1] which constantly store historical snapshots of the web page. *Dataset2* contains

[1] `https://archive.org/`

90 web pages (30 web pages and 60 page variants with a time interval of 1~77 months) from 22 popular websites covering 8 vertical fields such as news, travel agent and so on. Because the internet archive cannot collect everything from every website, when we collect web page and its page variants, only if the web page carrying the target information exists in the archive will be selected and saved. Therefore, the integrity of the page in Dataset2 is guaranteed. We note that the *Dataset2* only consists of pages from which the initial evaluated XPath failed to extract the target data in page variants. Furthermore, as best we know, there are very few unified approaches that suit both the detail page and the list page together. Therefore, the stability test on this data set will be more challenging. The *Dataset2* is an additional experiment. The reason we additionally prepare it is that: an XPath expression used on a site may receive a good score because the page does not change significantly and frequently. Because all original evaluated XPath extraction failed, this dataset can also be seen as the "recover rate" of the method.

As we mentioned in Section 1.2, mainly there are two reasons cause page variants: (1) inconsistent template, (2) internal structure change over time. At the same time or for a short time interval, most of the page variants are caused by the first reason, i.e., inconsistent template. Such page variants usually have minor structural changes that are either close to the leaves of the tree or just edited(insert/delete/replace) a small number of nodes. *Dataset1* consists mainly of such pages. While as the increase of time intervals, the internal structure of some page variants has changed largely (modifications close to the root of the tree; reorganization of the page; many nodes edited). *Dataset2* contains such temporal type of such pages. We use the "mixed" type change to describe the combination of the above two changes.

Therefore, the *Dataset2* is used to answer the 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 7.2: **The Composition of the *Dataset2* and its Result Record**

website	Initial Page 1		Page Variant 1-2						Page Variant 2-3						Page Variant 1-3						Variant Type	Page Type
	Date	Target Data	Interval	TED	Fire	Robular	Robular+	SSN/SSP/SSNP	Interval	TED	Fire	Robular	Robular+	SSN/SSP/SSNP	Interval	TED	Fire	Robular	Robular+	SSN/SSP/SSNP		
CNN Economy	20140101	top story	15m	0.427	✓	✓	×	× ✓ ✓	18m	0.461	×	×	×	✓ × ✓	32m	0.494	×	×	×	× × ✓	temporal	detail
BBC Business	20150101	top story	3m	0.449	×	×	×	× $\underline{\times}$ ✓	17m	0.074	×	×	×	✓ × ✓	20m	0.461	×	×	×	× × ✓	temporal	detail
Yahoo! News	20141020	navigation bar	18m	0.504	×	×	×	✓ ✓ ✓	3m	0.331	×	×	×	✓ × ✓	21m	0.516	×	×	×	× ✓ ✓	temporal	detail
booking.com	20151120	room score	4m	0.146	✓	×	×	✓ ✓ ✓	4m	0.209	✓	✓	✓	✓ ✓ ✓	8m	0.289	✓	×	×	✓ ✓ ✓	mix	detail
Ctrip Cruise	20150721	first title	10m	0.236	×	×	×	× $\underline{\times}$ ✓	2m	0.075	✓	×	×	✓ × ✓	12m	0.231	×	×	×	× $\underline{\times}$ ✓	mix	list
Lvmama Cruise	20151118	product title	6m	0.319	×	✓	✓	✓ ✓ ✓	1m	0.384	×	×	×	× × ✓	7m	0.377	×	×	×	× $\underline{\checkmark}$ ✓	mix	detail
Lvmama Tour	20160603	product title	2m	0.435	×	×	×	× × ×	10m	0.307	×	×	×	✓ $\underline{\times}$ ✓	12m	0.459	×	×	×	× × ×	mix	list
Ly Tour	20160229	product title	2m	0.070	✓	✓	×	× ✓ ✓	10m	0.276	✓	×	×	× × $\underline{\times}$	12m	0.263	✓	×	×	× × $\underline{\times}$	mix	detail
Ly Drive	20160602	product title	2m	0.174	×	×	×	✓ ✓ ✓	6m	0.260	✓	✓	✓	✓ ✓ ✓	8m	0.329	×	×	×	✓ × ✓	mix	detail
Ly Visa	20160225	product title	1m	0.387	×	×	×	× × ✓	3m	0.366	✓	✓	×	× ✓ ✓	4m	0.355	×	×	×	× × ×	mix	detail
	20160225	location	1m	0.387	×	×	×	× $\underline{\times}$ ✓	3m	0.366	×	×	✓	✓ ✓ ✓	4m	0.333	×	×	×	× $\underline{\times}$ $\underline{\checkmark}$	mix	detail
worldbank.org	20141012	GDP value	5m	0.066	✓	✓	×	✓ ✓ ✓	18m	0.408	×	×	×	✓ ✓ ✓	23m	0.415	×	×	×	✓ ✓ ✓	temporal	detail
wikipedia.org	20140614	first reference	9m	0.095	✓	✓	✓	✓ ✓ ✓	17m	0.187	×	✓	✓	✓ ✓ ✓	26m	0.244	×	×	×	× ✓ ✓	temporal	detail
	20140614	first note	9m	0.095	×	✓	✓	× × ✓	17m	0.187	×	✓	✓	✓ × ✓	26m	0.244	×	✓	✓	✓ ✓ ✓	temporal	detail
BBC *Country*	20040402	country map	20m	0.156	×	✓	✓	✓ ✓ ✓	57m	0.310	×	✓	✓	✓ ✓ ✓	77m	0.332	×	✓	✓	✓ ✓ ✓	temporal	detail
github.com	20140517	project name	9m	0.048	×	×	×	✓ ✓ ✓	18m	0.155	×	×	×	✓ ✓ ✓	27m	0.174	✓	×	×	✓ ✓ ✓	temporal	list
	20140517	company name	9m	0.048	✓	✓	✓	✓ ✓ ✓	18m	0.155	×	×	×	✓ ✓ ✓	27m	0.174	×	×	×	✓ ✓ ✓	temporal	list
	20140517	commit people	9m	0.048	✓	✓	✓	✓ ✓ ✓	18m	0.155	×	✓	✓	✓ ✓ ✓	27m	0.174	×	✓	✓	✓ ✓ ✓	temporal	list
zhihu.com	20151220	first subtopic	5m	0.187	×	×	✓	× ✓ ✓	4m	0.206	×	×	×	× × ×	9m	0.317	×	×	×	× × ×	temporal	list
amazon.co.jp	20160108	product price	7m	0.211	✓	✓	✓	× ✓ ✓	10m	0.211	✓	✓	✓	✓ ✓ ✓	17m	0.238	✓	✓	✓	✓ ✓ ✓	temporal	detail
	20160108	total price	7m	0.211	✓	✓	×	✓ ✓ ✓	10m	0.211	×	×	×	✓ ✓ ✓	17m	0.238	×	×	×	✓ ✓ ✓	temporal	detail
	20160108	detail info.	7m	0.211	×	×	✓	✓ ✓ ✓	10m	0.211	✓	×	✓	✓ ✓ ✓	17m	0.238	×	×	✓	✓ ✓ ✓	temporal	detail
ebay.com	20151216	first record	4m	0.113	×	×	×	✓ ✓ ✓	4m	0.300	×	×	×	× ✓ ✓	8m	0.308	×	×	×	× ✓ ✓	mix	list
	20151216	next page	4m	0.113	✓	✓	✓	✓ ✓ ✓	4m	0.300	✓	✓	✓	✓ ✓ ✓	8m	0.308	✓	✓	✓	✓ ✓ ✓	mix	list
youtube.com	20140617	video title	6m	0.302	✓	✓	✓	✓ ✓ ✓	21m	0.301	✓	✓	✓	✓ ✓ ✓	27m	0.378	✓	✓	✓	✓ ✓ ✓	mix	detail
	20140617	next video link	6m	0.302	×	×	×	✓ ✓ ×	21m	0.301	×	×	×	× ✓ ✓	27m	0.378	×	×	×	× × $\underline{\checkmark}$	mix	detail
eonline.com	20150108	video title	21m	0.283	✓	✓	✓	✓ ✓ ✓	16m	0.179	✓	✓	✓	✓ ✓ ✓	37m	0.324	✓	✓	✓	✓ ✓ ✓	mix	detail
csdn.net	20150102	article tag	15m	0.299	×	×	×	× ✓ ✓	4m	0.238	✓	×	×	× ✓ ✓	19m	0.250	×	×	×	× ✓ ✓	mix	detail
blog.sina.com	20140602	headline blog	16m	0.117	×	×	×	✓ $\underline{\checkmark}$ ✓	26m	0.189	×	×	×	✓ ✓ ✓	42m	0.225	×	×	×	✓ ✓ ✓	temporal	detail
twitter.com	20150318	first tweet	11m	0.200	×	×	×	✓ × ✓	8m	0.051	×	×	×	✓ ✓ ✓	19m	0.198	×	×	×	✓ × ✓	temporal	list

7.3 Experiment Environment

Different experiment environment, such as different HTML cleaning schemes, will convert page elements into different HTML DOM representation, which may lead to an inconsistent of XPath for the same target information. In order to make sure the experiments' result can be reproduced by other researchers we describe our experiment environment as follows. We implemented the prototype for our approaches with Java (JDK 1.7) whose main components are shown in Figure 6.2. All implementations and experiments are run on an Intel Core i5 2.4GHz PC running Windows 10 64bit with 8GB RAM.

Since most of visible contents of a web page are contained in the body element, instead of the <html> we regard the <body> node as the root node of the page. We use the *Jsoup*[90](ver.1.7.2) to traverse each HTML page into a post-ordered labeled tree, where each node of the tree is labeled with its tag name and has a unique order id. During the traversal, the tree is cleaned by deleting elements that have slight effect on contents recognition including formatting/style nodes (e.g. <hr>, <br>, <col>, etc.), external files nodes (e.g. ,<script>, etc.), empty/blank text nodes, image nodes with very small area, and the hidden input nodes. Full filtered nodes are as follows: NULL("Blank", "enter", "tab"), LINE BREAK("br"), LINK("link"), SCRIPT("script"), NOSCRIPT("noscript"), HR("hr"), STYLE("style"), DFN("dfn"), VAR("var"), CODE("code"), SAMP("samp"), KBD("kbd"), COL("col"), COLG("colgroup"), META("meta"), BASEF("basefont").

The XPath of the target node is generated by a plugin of the Firefox browser[2] named Firepath[3], which is an absolute or a relative path. The FirePath plugin was no longer supported by the Firefox version 51.0.1. We use the Firefox(version 30.0b1) and FirePath (version 0.9.7) to generate absolute XPath (abbreviated as XPath). We use the Firefox(version 49.0b9) and FirePath (version 0.9.7) for generated more relative paths (abbreviated as FirePath), which were mainly based on the id attribute. The Robular [67] and the Robular+ [29] robust XPath

[2]https://www.mozilla.org/en-US/firefox/new/

[3]https://addons.mozilla.org/en-us/firefox/addon/firepath/

locator (Robular and Robular+ for short) are publicly available at `http://sepl.dibris.unige.it/ROBULA.php`(accessed 15 March 2018). According to the authors, the Robular/Robular+ plugins were developed based on the FirePath (version 0.9.7) and worked on the Firefox before the version 31.0. The Robular/Robular+ were no longer supported since the Firefox version 34.

7.4 Experiment Process

The experiment process is shown in Figure 7.2. We first construct correct sets of goal nodes (ground truth) for two data sets. The ground truth (correct goal) of *Dataset1* was pre-defined and published by Qiang Hao et al.[21]. The correct goal nodes of *Dataset2* were screened by our volunteers, including 2 professional crawler developers and 4 college students having no web scraping experience. They first selected popular websites (22 websites of 8 categories) from the web traffic data ranking website: *www.alexa.com*. Then the crawler developers selected the *Target* based on the real extraction task experience in their work. For example, room score (*booking.com*), product price(*amazon.co.jp*), first product record content, next page link (*ebay.com*), video name, next video link (*youtube.com*) and etc. College students selected the target nodes based on their personal interests. For example, GDP value (*worldbank.org*), first reference, note (*wikipedia.org*).

For each page of *Dataset1*, there 3~5 target nodes are specified, and for each target node of a web page, 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 neighbor zone) is the version which directly uses the center node of the neighbor zone as the goal node. It is only the output of the Chapter 4.

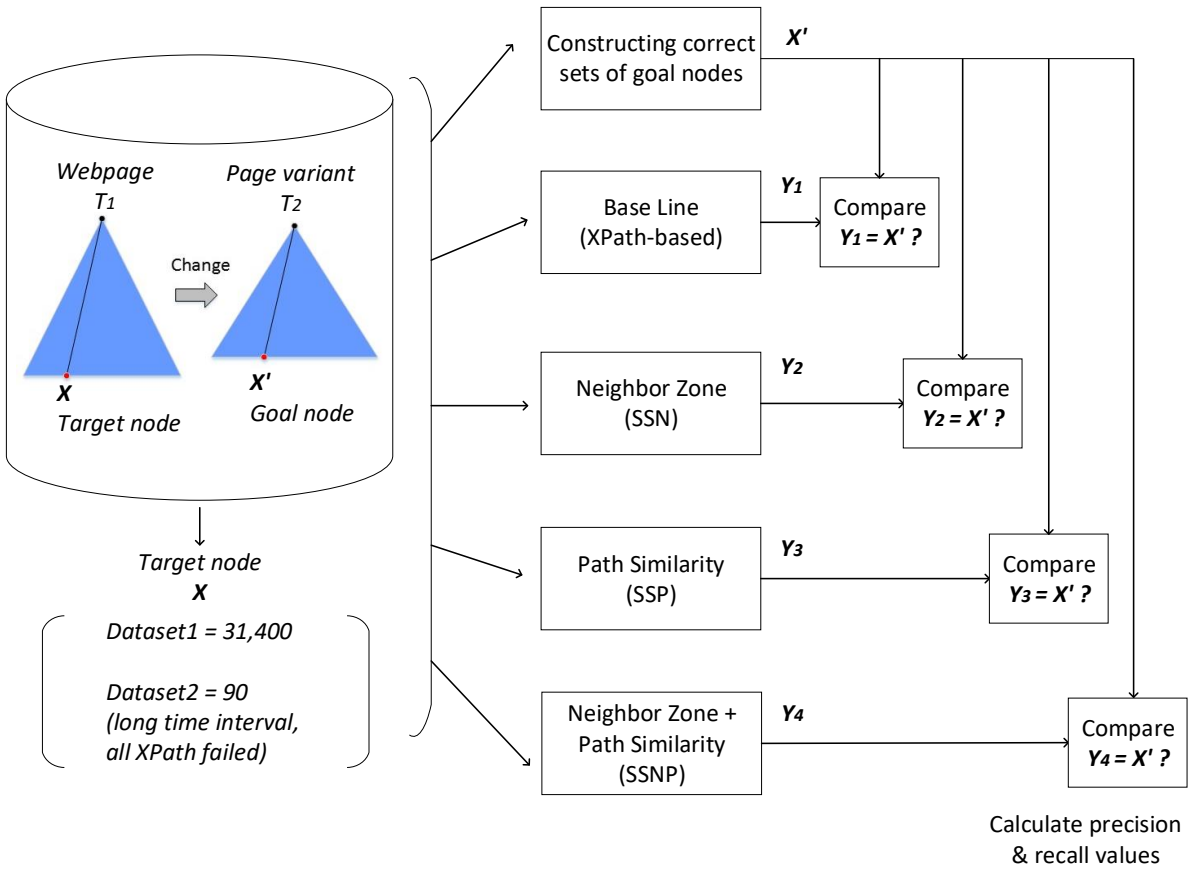


Figure 7.2: **Experiment Process**

- **SSP** (stable scraping based on path similarity) is the version which uses top ranking node(s) of all leaf nodes as the goal node. It is only the output of the Chapter 5.
- **SSNP** (stable scraping based on neighbor zone and path similarity) is the hybrid version which searches the goal node in nodes around the center node of neighbor zone using the path similarity. It actually is the SSN plus the SSP, which is the output of the Chapter 6.

7.5 Performance Metrics

Let a dataset be a finite set of web pages $W = \{w_1, ...w_i, ...w_n\}$; $X(w_{ij})$ 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 ($i = 1, ..., n$). For *Dataset1*, $n = 8000$ and $3 \leq k_i \leq 5$ ($i = 1, 2, ..., 8000$). For *Dataset2*, $n = 60$ and $1 \leq k_i \leq 3$ ($i = 1, 2, ..., 60$). Then the precision, recall and the F1-score are calculated by the formula as follows.

$$Precision = \frac{\sum_{i=1}^{n} \sum_{j=1}^{|k_i|} |X(w_{ij}) \cap Y(w_{ij})|}{\sum_{i=1}^{n} \sum_{j=1}^{|k_i|} |Y(w_{ij})|} \tag{7.1}$$

$$Recall = \frac{\sum_{i=1}^{n} \sum_{j=1}^{|k_i|} |X(w_{ij}) \cap Y(w_{ij})|}{\sum_{i=1}^{n} \sum_{j=1}^{|k_i|} |X(w_{ij})|} \tag{7.2}$$

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{7.3}$$

7.6 Experiment Results

In Figure 7.3 and 7.4, we summarize the precision, recall and F1-score of absolute XPath (XPath), relative XPath (FirePath), robular, robular+, SSN, SSP and SSNP in *DataSet1* and *Dataset2* respectively. The SSN, SSP method achieved a better overall result than XPath and robular/robular+, and the SSNP performs consistently better than other methods in the two datasets, which proves its stable effectiveness. Experiment results of vertical fields on *Dataset1* are shown in Figure 7.5.

Table 7.2 records all extraction results 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

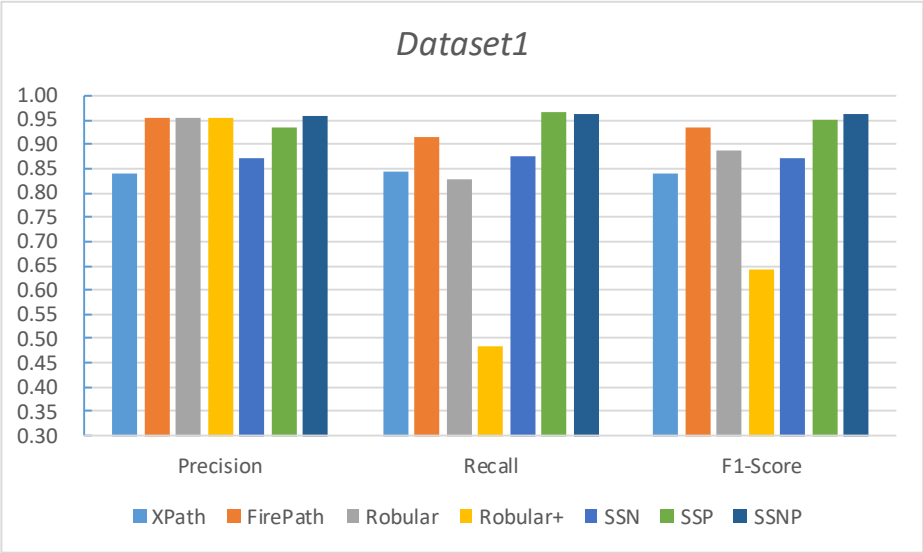


Figure 7.3: **The Precision, Recall and F1 Score of *Dataset1***

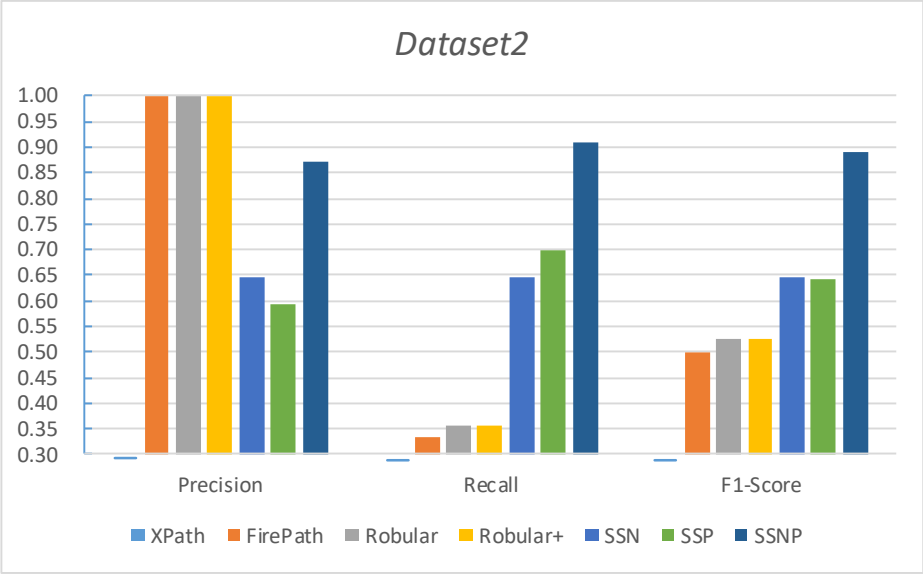


Figure 7.4: **The Precision, Recall and F1 Score of *Dataset2***

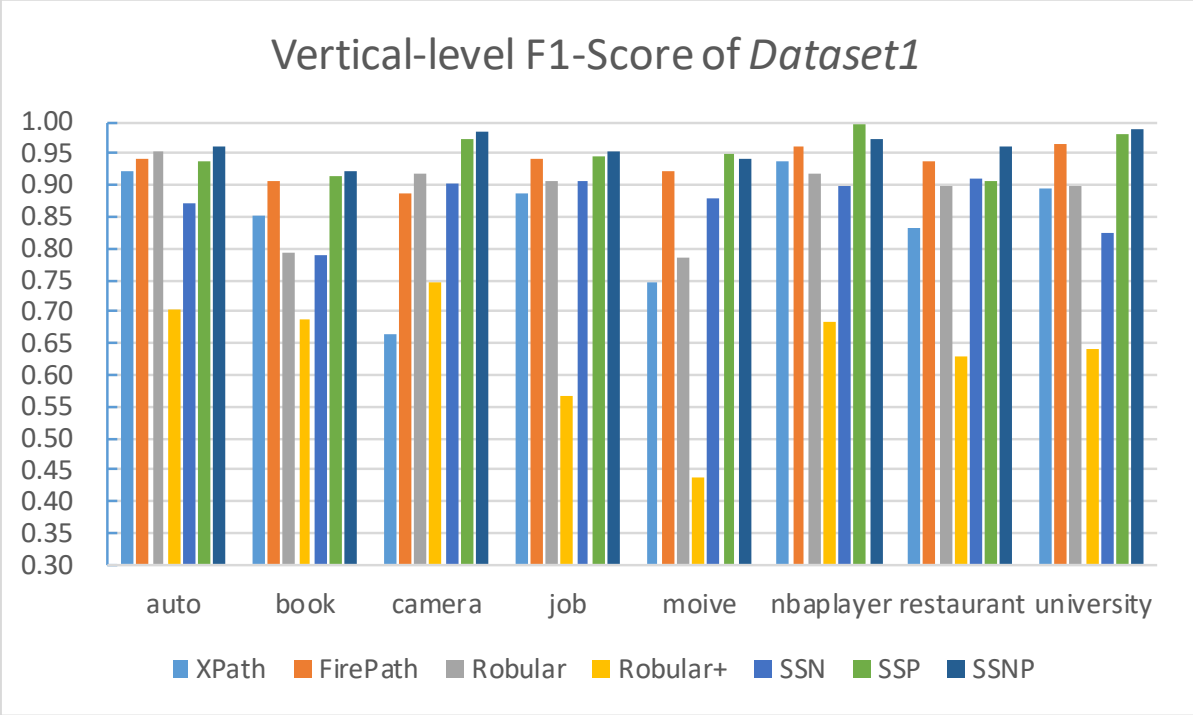


Figure 7.5: **The F1 Score of Vertical Field of *Dataset1***

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 temporal and mixed type page variants usually are more challenging for the extraction. 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 ✓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".

The detailed values of the above experimental results are shown in Table 7.3 and 7.4.

Table 7.3: **Experimental Results of Applying Different Methods to the Two Datasets**

Method	***Dataset1***			***Dataset2***		
	Precision	Recall	F1	Precision	Recall	F1
XPath	0.840	0.843	0.841	0	0	0
FirePath	0.954	0.915	0.934	1.000	0.333	0.500
Robular	0.953	0.828	0.886	1.000	0.356	0.525
Robular+	0.956	0.484	0.643	1.000	0.356	0.525
SSN	0.871	0.874	0.872	0.644	0.644	0.644
SSP	0.936	0.966	0.951	0.594	0.700	0.643
SSNP	0.957	0.964	0.960	0.872	0.911	0.891

Table 7.4: **Experimental Results of Applying Different Methods to the Dataset1 Verticals**

F1 Vertical / Method	auto	book	camera	job	moive	nbaplayer	restaurant	university
XPath	0.921	0.852	0.664	0.888	0.745	0.937	0.831	0.895
FirePath	0.943	0.907	0.888	0.942	0.924	0.961	0.937	0.967
Robular	0.952	0.792	0.920	0.905	0.787	0.918	0.900	0.898
Robular+	0.703	0.688	0.747	0.566	0.439	0.683	0.631	0.641
SSN	0.872	0.789	0.901	0.905	0.878	0.899	0.910	0.825
SSP	0.940	0.914	0.971	0.946	0.949	0.995	0.906	0.979
SSNP	0.963	0.922	0.983	0.953	0.942	0.973	0.962	0.987

7.7 Discussion

7.7.1 Answer to the Research Questions

From Figure 7.3, we see that in the *Dataset1* experiment, the absolute and relative XPath obtained an F1 score of 0.841 and 0.934 respectively. Correspondingly, the scores of SSN, SSP, SSNP are 0.872, 0.951 and 0.960. The SSN, SSP and SSNP methods increase the stability score of XPath by a maximum of 3.6%, 13.0%, 14.1% respectively. From Figure 7.4, because the *Dataset2* contains only page variants that the absolute XPath method failed to extract, the F1 score of absolute XPath is 0, relative XPath is 0.5, robular/robular+ both are 0.525, while SSN is 0.644, SSP is 0.643 and SSNP is 0.891. From these experimental results, we can see that, the SSN (averagely) and SSP methods have higher stability than absolute/relative XPath and robular/robular+ methods, and the SSNP method has more stable performance than all other methods. Therefore, for RQ1-1, there are two cases. When the page changes are small, such as a template variant, the SSN method extracts better results than absolute XPath, but is slightly weaker than id-based relative XPath, so the answer is *No*; when the page changes largely, such as a temporal or mix type page change, the SSN method performs better than both absolute and relative XPath, so the answer is *Yes*. The answers to the question RQ1-2 and RQ1-3 are *Yes*.

From Figure 7.4 and Table 7.2, we can see that as the time span increases, the page change grows largely. In this case, the XPath based extraction becomes fragile, while SSN, SSP reached the passing level (0.644, 0.643) and SSNP still maintains the F1-score above 0.89. Therefore, for the question RQ2, our answer is *Yes*. We note that all precision values of firepath, robular path and robular+ in the *Dataset2* are 1.0. The reason is that, for extracting a target node in a page variant, these methods either got the one correct result node or got none at all. For example, suppose that there are 100 extraction tasks (i.e., 100 target nodes), and the a method only extracted 40 result nodes which are all correct, then the precision = correct results /all extraction results = 40/40 = 1, the recall = correct results/ground truth = 40/100 = 0.4.

In conclusion, the proposed SSN, SSP, SSNP approaches have higher stability than the XPath in terms of extracting target nodes in page variants. In case of long time interval page variants, which are more difficult for the XPath method, the SSNP method still has a stable performance.

For other rival methods, the experimental results show that the Robular/Robular+ are more suitable for web application testing that focuses on the UI components oriented extraction, however, for extracting tasks that focus on content-oriented data in the page, their effectivenesses are not better than XPath and our methods. From Figure 7.5, we also found that our methods have higher stability than [21] in results of all the vertical fields (0.960 vs 0.844 average). We note the difference of application scenarios is that our methods need to specify a target node for each site without the help of visual files, while the [21] only needs to specify a target node for sites of a vertical field but requires visual information support.

In the rest of this section, we discuss the validity of our conclusions, the input requirements and the execution time of our approaches.

7.7.2 Observations from the Experiment Result

We have the following observations drawn from the 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, i.e., the future changed version of the template variant of the initial page, usually is more challenging for extraction.
- As shown in Figure 7.5, from experiment results of vertical fields on *Dataset1*, we found that few samples consisted of page variants caused by inconsistent templates in *Dataset1*,

where SSN, SSP have a better result than SSNP. For example, during the extraction of the camera price in the *jr.com* we get 0.92 F1-score by SSN and 0.72 by SSNP.

  - 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 5.4) to distinguish order sequences, for identifying the target data, we still need a more semantic level comparison.
  - The latter is because some web pages 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.

- What is the cause of the big gap between F1 score of SSN/SSP and SSNP for dataset2 in Figure 7.4? This actually verifies our observation in Section 1.4, which is a cornerstone of SSN method, i.e., "there are some page elements in a web page, and the relative positions between two of them remain unchanged or slightly changed in page variants". For the *Dataset2*, the effectiveness of SSN component has been challenged due to it is difficult to map two nodes exactly between the original page and the page variant which was largely changed. However, the *Goal* is still very probable in the neighbor zone. This means that even the center node of SSN was not the *Goal* directly, but it was very close to the *Goal* (within the radius r). Therefore, we developed the SSNP method that combines the SSN and SSP methods. In SSNP, the neighbor zone denotes an approximate location of the *Goal* in the page variant, which consists of a fragment of leaf nodes. And then we

search the precise location of the *Goal* by the ranking the path similarity score of nodes within the neighbor zone.

7.7.3 External Validity

We first discuss the threats to external validity (generalization) 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 popular websites having high hit counts covering 16 vertical areas, 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, next page link and 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 included.

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

7.7.4 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 customized 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.

- **Only HTML files.** The extractor should adapt to mainstream data rich web pages. Using only HTML files will reduce the bandwidth overhead for the big data collection task and can reduce the failure rate of the entire extraction process due to unsuccessful acquisition of other supplementary files like CSS or JavaScript files.

  - Missing the supplementary files (CSS, JavaScript) has limited impact on the extraction of target data. Without the supplementary files, i.e.CSS, JavaScript, some page layout, e.g., an advertisement block or interactive UI-components, e.g., a drop-down menu by mouse hover, of the page may be missing. However, we note that, firstly, our tasks focus on partially extract data from the main contents of web pages (not UI components oriented). Secondly, the HTML files the crawler/manually collected are the web pages that have already been rendered by scripts. Therefore, the pages still preserved many contents which had been dynamically generated by the scripts. Therefore, according to our observations, if the user could locate and select the target node (*Target*) in the original web page, then even if the subtree structure of the target node was generated by the dynamic scripts, the target node (*Goal*) would

still exist in the page variants.

- **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.

7.7.5 Execution Time

The running time of the the neighbor zone based method (SSN) is depended on the Zhang and Shasha TED algorithm[88]. We did not measure the time required by the SSN method because it largely depends on the implementation optimization. The theoretical computational complexity of the TED is as follows.

$$O(|T_1||T_2| \cdot min(leaves(T_1), height(T_1)) \cdot min(leaves(T_2), height(T_2))$$

We can see that it is acceptable in practical cases, because its running time depends on the height of the trees which usually is much lower compared to the number of all nodes or leaf nodes of the HTML tree. Some samples of this is shown in Table 7.5. Let the "#Nodes" column be the number of nodes in the page, "Height" the largest number of path layer of the HTML tree, "#Leaf Nodes" the leaf node number of the page.

The path similarity calculation between two paths is almost instantaneously. The execution time required by the path similarity based method is affected by the number of leaf nodes (one leaf node needs one calculation) in the HTML tree of page variants. For extracting the target node from 100 page variants, SSP method has required about 30 seconds on average. It denotes that, the whole extraction of a target node took only a very short time, i.e., 0.3 second on average, which can be considered well acceptable from the web scraping perspective of our experience. We also notice that as introduced in the implementation section Section 6.3, for the convenience of analysis, our methods have an initialization phase to store each node of HTML tree into the database. This accordingly will bring a loading phase for fetching the

Table 7.5: **The Number Of Nodes and the Height of the HTML Trees**

Website	Vertical	#Nodes	Height	#Leaf Nodes	Date
amazon.com	shopping	3528	25	1319	20170707
bbc.com	news	1759	20	573	20160916
booking.com	travel	5447	20	2630	20160408
worldbank.org	organization	1386	29	520	20160915
wikipedia.org	library	8790	22	3983	20160807
twitter.com	social networking	3594	25	1233	20160915
youtube.com	video	944	24	332	20160902
blog.sina.com	blog forum	1292	15	592	20161208

parsed HTML tree from the database which leads to redundant time consuming. Therefore, the execution time of our methods are acceptable for a real-world web extraction task.

Chapter 8

Conclusion

The aim of this thesis is to develop stable and flexible solutions for the partial information extraction on web pages without imposing complex restrictions on the input. The pattern matching and model learning based approaches need a learning phase or supplementary files. If the manually labelled sample set for learning is not sufficiently large, the extraction may be inaccurate. As a extraction agent, if the target data is frequently re-designated, the these methods will be inefficient.

In the thesis, we presented two partial information extraction methods and a hybrid method of them that focus on the stability problem caused by page variants. The first method, i.e., the neighbor zone based method, defines the concepts of the data type and the node distance, which describe the characteristics of page elements and the layout relationships between nodes in HTML trees. It presents the center node algorithm to search the location of the target node. The second method, i.e., the path similarity based method, presents an original approach to measure the similarity of page elements based on information carried by their XPaths. The path similarity between candidate nodes and the target node is evaluated from multiple aspects, i.e., tag path, attribute path, affiliation and list order, which is used to be rank.

In support of our conclusions, we experiment on two complementary datasets consist of real-world web pages. The first dataset contains 8,080 web pages from 80 websites of 8

vertical fields, where each page contains 3˜5 target nodes. Totally, we have extracted 31,400 target nodes from the *Dataset1*. The second dataset consists of 90 web pages from 22 popular websites of 8 vertical fields, where each page contains 1˜3 target nodes. It is more challenging because it only contains long time interval page variants that the XPath method failed to extract. The experimental results show that our methods have better stability for partical informatino extraction compared with the XPath method (F1-score increased by a maximum of 0.119 in Dataset1, 0.891 in Dataset2). By combining the neighbor zone(SSN) and the path similarity (SSP) methods, the SSNP gets a more stable extraction result (F1-score of SSN, SSP, SSNP: 0.872, 0.951, 0.960 in Dataset1, 0.644, 0.643, 0.891 in Dataset2). The presented approaches support arbitrary target data selecting in a page and returns the extraction result in time, because they only needs the HTML source files and do not need the pre-processing like labeling samples or designing patterns. This is significant because it complements existing XPath-based methods and is flexible for large-scale extraction.

In the future work, we think the following directions are important and interesting. We would like to further develop the calculation of unchanged node pairs and the semantic similarity of attributes to further improve the effectiveness of the SSN, SSP and SSNP methods. For example, we may optimize the mapping for HTML trees by developing a cost function that specifies a higher cost to the change operation in the TED algorithm. We are going to analyse the influence of different weight parameters in the tag path similarity functions in the real-world extraction tasks. We would like to construct a customizable scraping platform, which supports target choice through a GUI, e.g., a web browser plugin, 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.

References

[1] Y. Du, C. Su, Z. Cai, X. Guan, Web page and image semi-supervised classification with heterogeneous information fusion, Journal of Information Science 39 (3) (2013) 289–306. doi:10.1177/0165551513477818.

[2] E. Ferrara, P. D. Meo, G. Fiumara, R. Baumgartner, Web data extraction, applications and techniques: A survey, Knowledge-Based Systems 70 (Supplement C) (2014) 301–323. doi:10.1016/j.knosys.2014.07.007.

[3] S. A. Catanese, P. De Meo, E. Ferrara, G. Fiumara, A. Provetti, Crawling facebook for social network analysis purposes, in: Proceedings of the International Conference on Web Intelligence, Mining and Semantics, WIMS '11, ACM, 2011, pp. 52:1–52:8. doi:10.1145/1988688.1988749.

[4] W. contributors, Nate silver — wikipedia, the free encyclopedia, [Online; accessed 1-March-2018] (2018).
URL https://en.wikipedia.org/w/index.php?title=Nate_Silver&oldid=826207315

[5] J. Robie, D. Chamberlin, M. Dyck, J. Snelson, Xml path language (xpath) 3.0, https://www.w3.org/TR/xpath-30 (2014).

[6] J. Myllymaki, Effective web data extraction with standard xml technologies, Computer Networks 39 (5) (2002) 635–644. doi:10.1145/371920.372183.

[7] H. Liu, Y. X. Ma, Web data extraction research based on wrapper and xpath technology, in: Advanced Materials and Information Technology Processing, Vol. 271 of Advanced Materials Research, Trans Tech Publications, 2011, pp. 706–712. doi:10.4028/www.scientific.net/AMR.271-273.706.

[8] R. De Mol, A. Bronselaer, J. Nielandt, G. De Tré, Data driven xpath generation, in: Intelligent Systems' 2014. Advances in Intelligent Systems and Computing, vol 322., Springer,Cham, 2015, pp. 569–580.

[9] N. Dalvi, P. Bohannon, F. Sha, Robust web extraction: An approach based on a probabilistic tree-edit model, in: Proceedings of the 2009 ACM SIGMOD International Conference on Management of data, ACM, 2009, pp. 335–348.

[10] H. Han, T. Tokuda, A personal web information/knowledge retrieval system, Frontiers In Artificial Intelligence And Applications 166 (2008) 338–345.

[11] E. Adar, J. Teevan, S. T. Dumais, J. L. Elsas, The web changes everything: Understanding the dynamics of web content, in: Proceedings of the Second ACM International Conference on Web Search and Data Mining, 2009, pp. 282–291. doi:10.1145/1498759.1498837.

[12] P. H. Cording, K. Lyngby, Algorithms for web scraping, M.Sc. Thesis, Technical University of Denmark (2011).
URL http://www2.imm.dtu.dk/pubdb/views/publication_details.php?id=6183

[13] E. Ferrara, R. Baumgartner, Automatic wrapper adaptation by tree edit distance matching, in: Combinations of Intelligent Methods and Applications, Springer, 2011, pp. 41–54.

[14] J. Nielandt, R. De Mol, A. Bronselaer, G. De Tré, Wrapper induction by xpath alignment, in: Proceedings of the 6th International Conference on Knowledge Discovery and Information Retrieval, Vol. 6, Science and Technology Publications, 2014, pp. 492–500.

[15] J. P. Cohen, W. Ding, A. Bagherjeiran, Semi-supervised web wrapper repair via recursive tree matching, CoRR arXiv:1505.01303 [cs.IR]. arXiv:1505.01303.
URL http://arxiv.org/abs/1505.01303

[16] J. Nielandt, A. Bronselaer, G. de Tré, Predicate enrichment of aligned xpaths for wrapper induction, Expert Systems with Applications 51 (C) (2016) 259–275. doi:10.1016/j.eswa.2015.12.040.
URL http://dx.doi.org/10.1016/j.eswa.2015.12.040

[17] A. Parameswaran, N. Dalvi, H. Garcia-Molina, R. Rastogi, Optimal schemes for robust web extraction, in: Proceedings of the VLDB Conference, Vol. 4, 2011, pp. 980–991.

[18] T. Furche, J. Guo, S. Maneth, C. Schallhart, Robust and noise resistant wrapper induction, in: Proceedings of the 2016 International Conference on Management of Data, SIGMOD '16, ACM, New York, NY, USA, 2016, pp. 773–784. doi:10.1145/2882903.2915214.

[19] Y. Zhai, B. Liu, Web data extraction based on partial tree alignment, in: Proceedings of the 14th International Conference on World Wide Web, WWW '05, ACM, 2005, pp. 76–85. doi:10.1145/1060745.1060761.

[20] W. Liu, X. Meng, W. Meng, Vide: A vision-based approach for deep web data extraction, IEEE Transactions on Knowledge and Data Engineering 22 (3) (2010) 447–460. doi:10.1109/TKDE.2009.109.

[21] Q. Hao, R. Cai, Y. Pang, L. Zhang, From one tree to a forest: A unified solution for structured web data extraction, in: Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval, ACM, 2011, pp. 775–784.

[22] J. Zeng, F. Li, B. Flanagan, S. Hirokawa, Ltde: A layout tree based approach for deep page data extraction, IEICE Transactions on Information and Systems 100 (5) (2017) 1067–1078.

[23] M. Abe, M. Hori, Robust pointing by xpath language: Authoring support and empirical evaluation, in: Proceedings 2003 Symposium on Applications and the Internet, IEEE, 2003, pp. 156–165.

[24] M. Kowalkiewicz, T. Kaczmarek, W. Abramowicz, Myportal: Robust extraction and aggregation of web content, in: Proceedings of the 32nd international conference on very large data bases, VLDB Endowment, 2006, pp. 1219–1222.

[25] P. Gao, H. Han, T. Tokuda, Stable partial information extraction: A self-evolving hybrid mechanism, Frontiers in Artificial Intelligence and Applications 251 (2013) 49–62. doi:10.3233/978-1-61499-177-9-49.
URL 10.3233/978-1-61499-177-9-49

[26] D. C. Reis, P. B. Golgher, A. S. Silva, A. Laender, Automatic web news extraction using tree edit distance, in: Proceedings of the 13th international conference on World Wide Web, ACM, 2004, pp. 502–511.

[27] A. Omari, S. Shoham, E. Yahav, Synthesis of forgiving data extractors, in: Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, ACM, 2017, pp. 385–394.

[28] T. Furche, G. Gottlob, G. Grasso, C. Schallhart, A. Sellers, Oxpath: A language for scalable data extraction, automation, and crawling on the deep web, The VLDB Journal 22 (1) (2013) 47–72. doi:10.1007/s00778-012-0286-6.

[29] M. Leotta, A. Stocco, F. Ricca, P. Tonella, Robula+: an algorithm for generating robust xpath locators for web testing, Journal of Software: Evolution and Process 28 (3) (2016) 177–204, jSME-15-0035.R1. doi:10.1002/smr.1771.
URL 10.1002/smr.1771

[30] J. Guo, H. Han, A method for facilitating end-user mashup based on description, International Journal of Web Engineering and Technology 9 (2) (2014) 99–124.

[31] J. H. Liu, L. U. Yu-Liang, Survey on topic-focused web crawler, Application Research of Computers 24 (10) (2007) 26–29.

[32] T. Peng, L. Liu, Clustering-based topical web crawling for topic-specific information retrieval guided by incremental classifier, International Journal of Software Engineering and Knowledge Engineering 25 (01) (2015) 147–168.

[33] F. Zhao, J. Zhou, C. Nie, H. Huang, H. Jin, Smartcrawler: A two-stage crawler for efficiently harvesting deep-web interfaces, IEEE transactions on services computing 9 (4) (2016) 608–620. doi:10.1109/TSC.2015.2414931.

[34] C. Behnert, D. Lewandowski, A framework for designing retrieval effectiveness studies of library information systems using human relevance assessments, Journal of Documentation 73 (3) (2017) 509–527.

[35] R. L. Tulasi, M. S. Rao, K. Ankita, R. Hgoudar, Ontology-based automatic annotation: An approach for efficient retrieval of semantic results of web documents, in: Proceedings of the First International Conference on Computational Intelligence and Informatics, Springer, 2017, pp. 331–339.

[36] M. Mintz, S. Bills, R. Snow, D. Jurafsky, Distant supervision for relation extraction without labeled data, in: Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2, Association for Computational Linguistics, 2009, pp. 1003–1011.

[37] F. Wu, D. S. Weld, Open information extraction using wikipedia, in: Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, 2010, pp. 118–127.

[38] A. Ritter, L. Zettlemoyer, O. Etzioni, et al., Modeling missing data in distant supervision

for information extraction, Transactions of the Association for Computational Linguistics 1 (2013) 367–378.

[39] J. Hammer, J. McHugh, H. Garcia-Molin, Semistructured data: The tsimmis experience, in: Proceedings of the First East-European Conference on Advances in Databases and Information Systems, ADBIS'97, BCS Learning & Development Ltd., Swindon, UK, 1997, pp. 22–22.
URL `http://dl.acm.org/citation.cfm?id=2227663.2227685`

[40] G. O. Arocena, A. O. Mendelzon, Weboql: Restructuring documents, databases and webs, in: Data Engineering, 1998. Proceedings., 14th International Conference on, IEEE, 1998, pp. 24–33.

[41] V. Crescenzi, G. Mecca, Grammars have exceptions, Information Systems 23 (8) (1998) 539–565.

[42] W. Gatterbauer, P. Bohunsky, M. Herzog, B. Krüpl, B. Pollak, Towards domain-independent information extraction from web tables, in: Proceedings of the 16th international conference on World Wide Web, ACM, 2007, pp. 71–80.

[43] T. Furche, G. Gottlob, G. Grasso, X. Guo, G. Orsi, C. Schallhart, Opal: Automated form understanding for the deep web, in: Proceedings of the 21st international conference on World Wide Web, ACM, 2012, pp. 829–838.

[44] D. Pinto, A. McCallum, X. Wei, W. B. Croft, Table extraction using conditional random fields, in: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval, ACM, 2003, pp. 235–242.

[45] Z. Chen, M. Cafarella, Automatic web spreadsheet data extraction, in: Proceedings of the 3rd International Workshop on Semantic Search over the Web, ACM, 2013, pp. 1–8. `doi:10.1145/2509908.2509909`.

[46] M. K. Bergman, White paper: the deep web: surfacing hidden value, Journal of electronic publishing 7 (1).

[47] T. Furche, G. Gottlob, G. Grasso, X. Guo, G. Orsi, C. Schallhart, OPAL: Automated form understanding for the deep web, in: Proc. of the 21st World Wide Web Conf. (WWW), 2012, pp. 829–838. doi:10.1145/2187836.2187948.
URL http://dl.acm.org/citation.cfm?doid=2187836.2187948

[48] W. Gatterbauer, P. Bohunsky, M. Herzog, B. Krüpl, B. Pollak, Towards domain-independent information extraction from web tables, in: Proceedings of the 16th International Conference on World Wide Web, WWW '07, ACM, New York, NY, USA, 2007, pp. 71–80. doi:10.1145/1242572.1242583.
URL http://doi.acm.org/10.1145/1242572.1242583

[49] G. Miao, J. Tatemura, W.-P. Hsiung, A. Sawires, L. E. Moser, Extracting data records from the web using tag path clustering, in: Proceedings of the 18th International Conference on World Wide Web, WWW '09, ACM, New York, NY, USA, 2009, pp. 981–990. doi:10.1145/1526709.1526841.
URL http://doi.acm.org/10.1145/1526709.1526841

[50] G. Miao, J. Tatemura, W.-P. Hsiung, A. Sawires, L. E. Moser, Extracting data records from the web using tag path clustering, in: Proceedings of the 18th International Conference on World Wide Web, ACM, 2009, pp. 981–990.

[51] Y. Fang, X. Xie, X. Zhang, R. Cheng, Z. Zhang, Stem: A suffix tree-based method for web data records extraction, Knowledge and Information Systems (2017) 1–27doi:10.1007/s10115-017-1062-0.

[52] L. N. L. Figueiredo, G. T. de Assis, A. A. Ferreira, Derin: A data extraction method based on rendering information and n-gram, Information Processing & Management 53 (5) (2017) 1120–1138. doi:10.1016/j.ipm.2017.04.007.

[53] H. Han, T. Tokuda, A layout-independent web news article contents extraction method based on relevance analysis, Web Engineering (2009) 453–460doi:10.1007/978-3-642-02818-2_37.

[54] S. Zheng, R. Song, J.-R. Wen, Template-independent news extraction based on visual consistency, in: Proceedings of the 22Nd National Conference on Artificial Intelligence - Volume 2, AAAI'07, AAAI Press, 2007, pp. 1507–1512.

[55] S. Wu, J. Liu, J. Fan, Automatic web content extraction by combination of learning and grouping, in: Proceedings of the 24th International Conference on World Wide Web, International World Wide Web Conferences Steering Committee, 2015, pp. 1264–1274.

[56] Q. Wang, Q. Yang, J. Zhang, R. Zhou, Y. Zhang, Extracting web content by exploiting multi-category characteristics, in: Proceedings of International Conference on Web Information Systems Engineering, Springer, 2017, pp. 229–244.

[57] J. Wang, C. Chen, C. Wang, J. Pei, J. Bu, Z. Guan, W. V. Zhang, Can we learn a template-independent wrapper for news article extraction from a single training site?, in: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2009, pp. 1345–1354.

[58] B. Audeh, M. Beigbeder, A. Zimmermann, P. Jaillon, C. Bousquet, Vigi4med scraper: A framework for web forum structured data extraction and semantic representation, PLOS ONE 12 (1) (2017) 1–18. doi:10.1371/journal.pone.0169658.
URL https://doi.org/10.1371/journal.pone.0169658

[59] D. L. Sánchez, J. Revuelta, F. De la Prieta, A. B. Gil-González, C. Dang, Twitter user clustering based on their preferences and the louvain algorithm, in: Trends in Practical Applications of Scalable Multi-Agent Systems, the PAAMS Collection, Springer, 2016, pp. 349–356.

[60] J. Wong, J. Hong, Making mashups with marmite: towards end-user programming for the web, in: Proceedings of the SIGCHI conference on Human factors in computing systems, ACM, 2007, pp. 1435–1444.

[61] T. Suzuki, T. Tokuda, Path set operations for clipping of parts of web pages and information extraction from web pages, in: Proceedings of the 15th International Conference on Software Engineering and Knowledge Engineering, Citeseer, 2003, pp. 547–554.

[62] S. Gupta, G. Kaiser, Extracting content from accessible web pages, in: Proceedings of the 2005 International Cross-Disciplinary Workshop on Web Accessibility (W4A), May, Citeseer, 2005, pp. 10–10.

[63] Yahoo, Dapper, `http://open.dapper.net/`.

[64] H. Han, T. Tokuda, Wike: A web information/knowledge extraction system for web service generation, in: Proceedings of the 8th International Conference on Web Engineering, 2008, pp. 354–357.

[65] S. Lingam, S. Elbaum, Supporting end-users in the creation of dependable web clips, in: Proceedings of the 16th international conference on World Wide Web, ACM, 2007, pp. 953–962.

[66] I. n. Paz, O. Díaz, Providing resilient xpaths for external adaptation engines, in: Proceedings of the 21st ACM Conference on Hypertext and Hypermedia, HT '10, ACM, New York, NY, USA, 2010, pp. 67–76. `doi:10.1145/1810617.1810631`.
URL `http://doi.acm.org/10.1145/1810617.1810631`

[67] M. Leotta, A. Stocco, F. Ricca, P. Tonella, Reducing web test cases aging by means of robust xpath locators, in: Software Reliability Engineering Workshops (ISSREW), 2014 IEEE International Symposium on, IEEE, 2014, pp. 449–454.

[68] P. Van Hai, T. Aoyagi, T. Noro, T. Tokuda, Towards automatic detection of potentially important international events/phenomena from news articles at mostly domestic news sites, in: Proceeding of the 2007 conference on Information Modelling and Knowledge Bases XVIII, IOS Press, 2007, pp. 277–284.

[69] E. Ferrara, R. Baumgartner, Intelligent self-repairable web wrappers, in: Congress of the Italian Association for Artificial Intelligence, Springer, 2011, pp. 274–285.

[70] E. Ferrara, R. Baumgartner, Design of automatically adaptable web wrappers, in: Proceeding of the 3rd International Conference on Agents and Artificial Intelligence, 2011, pp. 211–217.

[71] Y. Kim, J. Park, T. Kim, J. Choi, Web information extraction by html tree edit distance matching, in: Convergence Information Technology, 2007. International Conference on, IEEE, 2007, pp. 2455–2460.

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

[73] Z. Zhang, C. Zhang, Z. Lin, B. Xiao, Blog extraction with template-independent wrapper, in: Network Infrastructure and Digital Content, 2010 2nd IEEE International Conference on, IEEE, 2010, pp. 313–317.

[74] M. E. Califf, R. J. Mooney, Bottom-up relational learning of pattern matching rules for information extraction, Journal of Machine Learning Research 4 (Jun) (2003) 177–210.

[75] N. Kushmerick, Wrapper induction: Efficiency and expressiveness, Artificial Intelligence 118 (1-2) (2000) 15–68.

[76] S. Soderland, Learning information extraction rules for semi-structured and free text, Machine learning 34 (1-3) (1999) 233–272.

[77] D. Freitag, Machine learning for information extraction in informal domains, Machine learning 39 (2-3) (2000) 169–202.

[78] C.-N. Hsu, M.-T. Dung, Generating finite-state transducers for semi-structured data extraction from the web, Information systems 23 (8) (1998) 521–538.

[79] I. Muslea, S. Minton, C. Knoblock, A hierarchical approach to wrapper induction, in: Proceedings of the third annual conference on Autonomous Agents, ACM, 1999, pp. 190–197.

[80] X.-H. Phan, S. Horiguchi, T.-B. Ho, Automated data extraction from the web with conditional models, International Journal of Business Intelligence and Data Mining 1 (2) (2005) 194–209.

[81] J. Turmo, A. Ageno, N. Català, Adaptive information extraction, ACM Computing Surveys (CSUR) 38 (2) (2006) 4.

[82] N. Dalvi, R. Kumar, M. Soliman, Automatic wrappers for large scale web extraction, Proceedings of the VLDB Endowment 4 (4) (2011) 219–230.

[83] D. López-Sánchez, J. M. Corchado, A. G. Arrieta, A cbr system for image-based webpage classification: Case representation with convolutional neural networks (2017).

[84] D. López-Sánchez, A. G. Arrieta, J. M. Corchado, Deep neural networks and transfer learning applied to multimedia web mining, in: International Symposium on Distributed Computing and Artificial Intelligence, Springer, 2017, pp. 124–131.

[85] M. Stanton, T. Hartley, F. Loizides, A. Worrallo, Dual-mode user interfaces for web based interactive 3d virtual environments using three. js, in: IFIP Conference on Human-Computer Interaction, Springer, 2017, pp. 441–444.

[86] J. Marini, Document object model, McGraw-Hill, Inc., 2002.

[87] K.-C. Tai, The tree-to-tree correction problem, Journal of the ACM (JACM) 26 (3) (1979) 422–433.

[88] K. Zhang, D. Shasha, Simple fast algorithms for the editing distance between trees and related problems, SIAM journal on computing 18 (6) (1989) 1245–1262. doi:10.1137/0218082.

[89] R. A. Wagner, M. J. Fischer, The string-to-string correction problem, Journal of the ACM (JACM) 21 (1) (1974) 168–173. doi:10.1145/321796.321811.

[90] J. Hedley, et al., jsoup: Java html parser, 2017, Website (https://jsoup.org/).