VILNIUS GEDIMINAS TECHNICAL UNIVERSITY

Tomas GRIGALIS

STRUCTURED DATA EXTRACTION FROM TEMPLATE-GENERATED WEB PAGES

DOCTORAL DISSERTATION

TECHNOLOGICAL SCIENCES,
INFORMATICS ENGINEERING (07T)

VGTU
LEIDYKLA
TECHNIKA
Vilnius 2014

Doctoral dissertation was prepared at Vilnius Gediminas Technical University in 2010–2014.

Supervisor

Prof Dr Habil Antanas ČENYS (Vilnius Gediminas Technical University, Informatics Engineering – 07T).

The Dissertation Defense Council of Scientific Field of Informatics Engineering of Vilnius Gediminas Technical University:

Chairman

Assoc Prof Dr Arnas KAČENIAUSKAS (Vilnius Gediminas Technical University, Informatics Engineering – 07T).

Members:

Dr Robertas DAMAŠEVIČIUS (Kaunas University of Technology, Informatics Engineering – 07T),

Prof Dr Habil Gintautas DZEMYDA (Vilnius University, Informatics Engineering – 07T),

Dr Mario Rosario GUARRACINO (Institute for High Performance Computing and Networking, Italy, Mathematics – 01P),

Prof Dr Olegas VASILECAS (Vilnius Gediminas Technical University, Informatics Engineering – 07T).

The dissertation will be defended at the public meeting of the Dissertation Defense Council of Informatics Engineering in the Senate Hall of Vilnius Gediminas Technical University at 10 a. m. on 16 September 2014.

Address: Saulėtekio al. 11, LT-10223 Vilnius, Lithuania.
Tel.: +370 5 274 4956; fax +370 5 270 0112; e-mail: doktor@vgtu.lt

A notification on the intend defending of the dissertation was send on 14 August 2014.

A copy of the doctoral dissertation is available for review at the Internet website http://dspace.vgtu.lt and at the Library of Vilnius Gediminas Technical University (Saulėtekio al. 14, LT-10223 Vilnius, Lithuania).

VGTU leidyklos TECHNIKA 2262-M mokslo literatūros knyga

ISBN 978-609-457-699-7

© VGTU leidykla TECHNIKA, 2014
© Tomas Grigalis, 2014
tomas.grigalis@vgtu.lt

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

Tomas GRIGALIS

STRUKTŪRIZUOTŲ DUOMENŲ IŠGAVIMAS IŠ TINKLALAPIŲ SUGENERUOTŲ PAGAL ŠABLONUS

DAKTARO DISERTACIJA

TECHNOLOGIJOS MOKSLAI,
INFORMATIKOS INŽINERIJA (07T)

VGTU
LEIDYKLA
TECHNIKA
Vilnius 2014

Disertacija rengta 2010–2014 metais Vilniaus Gedimino technikos universitete.

Vadovas

prof. habil. dr. Antanas ČENYS (Vilniaus Gedimino technikos universitetas, informatikos inžinerija – 07T).

Vilniaus Gedimino technikos universiteto Informatikos inžinerijos mokslo krypties disertacijos gynimo taryba:

Pirmininkas

doc. dr. Arnas KAČENIAUSKAS (Vilniaus Gedimino technikos universitetas, informatikos inžinerija – 07T).

Nariai:

dr. Robertas DAMAŠEVIČIUS (Kauno technologijos universitetas, informatikos inžinerija – 07T),
prof. habil. dr. Gintautas DZEMYDA (Vilniaus universitetas, informatikos inžinerija – 07T),
dr. Mario Rosario GUARRACINO (Didelės spartos skaičiavimų ir tinklų institutas, Italija, matematika – 01P),
prof. dr. Olegas VASILECAS (Vilniaus Gedimino technikos universitetas, informatikos inžinerija – 07T).

Disertacija bus ginama viešame Informatikos inžinerijos mokslo krypties disertacijos gynimo tarybos posėdyje 2014 m. rugsėjo 16 d. 10 val. Vilniaus Gedimino technikos universiteto senato posėdžių salėje.

Adresas: Saulėtekio al. 11, LT-10223 Vilnius, Lietuva.
Tel.: (8 5) 274 4956; faksas (8 5) 270 0112; el. paštas doktor@vgtu.lt

Pranešimai apie numatomą ginti disertaciją išsiųsti 2014 m. rugpjūčio 14 d.

Disertaciją galima peržiūrėti interneto svetainėje http://dspace.vgtu.lt/ ir Vilniaus Gedimino technikos universiteto bibliotekoje (Saulėtekio al. 14, LT-10223 Vilnius, Lietuva).

Abstract

Most of structured data on the Web is found in database-backed web sites. Typically, upon a web page request in such a site, structured data is retrieved from an underlying database and embedded into a web page using some fixed template. Reverse engineering task – extracting structured data from template-generated web pages is studied in this dissertation. There are thousands of web pages on the Web that differ in visual style and underlying structure. Automatically extracting structured data from many structurally heterogonous template-generated web pages is a difficult and time consuming task, and it is regarded as a grand challenge. It is assumed, that solving the challenge would improve todays' Web search and help companies to reduce costs. Thus the main goal of the dissertation is to propose a novel and more effective method for extracting structured data from template-generated web pages. The object of the research in this dissertation is structured data extraction from template-generated web pages.

The dissertation consists of introduction, four main chapters and general conclusions. In the first Chapter the problem of structured web data extraction is introduced, state-of-the-art data extraction techniques are reviewed and finally real life applications for structured web data extraction systems are discussed.

In the second Chapter a novel method for extracting structured data records from template-generated web pages is presented. The method is based on clustering visually and structurally similar web page elements. It first renders a given web page in a contemporary web browser, and then clusters visually and structurally similar repeating web page elements to identify an underlying pattern of embedded structured data records. Finally a data extracting wrapper is generated. The wrapper consists of XPath expressions that can be easily reused in many third party data extracting applications.

In the third Chapter a novel method for structurally clustering template-generated web pages is proposed. The method is based on the three observations: that there is a limited number of different style templates in one particular template-generated web site; that there is a limited number of inner-site link locations in all templates of a same site; that each individual location in a web page containing a link usually points to structurally similar web pages. The method leverages XPath locations of inbound inner-site links to significantly speed up web page clustering time.

In the final fourth Chapter more than one million web pages are used to experimentally evaluate the two proposed methods. The results reveal that the both proposed methods consistently outperform other state-of-the-art techniques.

Reziumė

Dauguma struktūrizuotų duomenų internete yra randami duomenų bazėmis paremtose interneto svetainėse. Paprastai, naršant tokio tipo svetainėse, kiekvienos užklausos metu yra kreipiamasi į duomenų bazę ir iš jos ištraukiami struktūrizuoti duomenys. Naudojant iš anksto paruoštus šablonus šie duomenys yra automatiškai integruojami į naršomą tinklalapį ir atvaizduojami vartotojui. Šioje disertacijoje yra tyrinėjama kaip šiuos duomenų išgauti iš minėtų tinklalapių. Internete gausu skirtingo dizaino ir struktūros internetinių svetainių, todėl siekis automatiškai atpažinti nežinomos struktūros tinklalapius ir išgauti juose esančius struktūrizuotus duomenis yra itin sudėtinga problema. Manoma, jog išsprendus šią problemą būtų galima pagerinti informacijos paieškos internete sistemas ir įgalinti organizacijas žymiai sumažinti internetinių duomenų rinkimo kaštus. Tad šios disertacijos tikslas yra pasiūlyti naują ir efektyvesnį metodą, skirtą išgauti struktūrizuotus duomenis iš tinklalapių sugeneruotų pagal šablonus. Disertacijos tyrimų objektas – struktūrizuotų duomenų išgavimas iš tinklalapių sugeneruotų pagal šablonus.

Disertaciją sudaro įvadas, keturi pagrindiniai skyriai ir bendrosios išvados. Pirmajame skyriuje yra supažindinama su struktūrizuotų duomenų gavybos internete problema, nagrinėjami pažangiausi struktūrizuotų duomenų gavybos metodai, jų pritaikymas verslo analitikos sistemose.

Antrajame skyriuje pristatomas naujas metodas skirtas automatiškai išgauti struktūrizuotus duomenis iš tinklalapių sugeneruotų pagal šablonus. Metodas yra grįstas struktūriškai ir vizualiai panašių tinklalapio elementų klasterizacija. Vaizdinei informacijai išgauti tinklalapis yra atvaizduojamas interneto naršyklėje. Vizualiai ir struktūriškai panašūs tinklalapio elementai suklasterizuojami, o gauti klasteriai išnaudojami siekiant atpažinti tinklalapyje esančių duomenų struktūrą. Tuomet automatiškai sugeneruojamas XPath eilučių rinkinys, kurį naudojant galima išgauti struktūrizuotus duomenis iš atitinkamo dizaino tinklalapio.

Trečiajame skyriuje pristatomas naujas metodas skirtas sparčiai klasterizuoti panašios struktūros tinklalapius. Metodas grįstas trimis įžvalgomis: kad vienoje interneto svetainėje dažniausiai yra randamas ribotas skaičius skirtingo dizaino vidinių tinklalapių; kad kiekvienam svetainės tinklalapio šablone yra numatytas ribotas skaičius vietų, kuriose yra talpinamos nuorodos į kitus vidinius svetainės tinklalapius; kad kiekviename to paties šablono tinklalapyje konkreti vieta su nuoroda veda į tarpusavyje panašios struktūros tinklalapius.

Paskutiniame ketvirtajame skyriuje yra eksperimentiškai išbandomi pasiūlyti metodai naudojant daugiau kaip vieną milijoną tinklalapių. Bandymų rezultatai atskleidžia, jog abu pasiūlyti metodai visais išbandytais atvejais savo efektyvumu lenkia kitų autorių naujausius metodus.

Notations

Symbols

$\|DataItems\|$	–	number of data items per given data record
$\|N_1\|, \|N_2\|$	–	number of nodes in corresponding HTML trees
$\|s(D_n)\|$	–	the size of a shingle set from a given document D_n
$\|s_1\|, \|s_2\|$	–	length of the strings
$\|xp(P_i)\|$	–	size of a given XPaths set
A,B	–	HTML trees
c	–	containment of given text documents
CPD	–	common paths distance of given paths sets
d	–	string edit distance
D_1, D_2	–	text documents
DR_a	–	area in pixels of a data region in a web page
$DR_{actural}$	–	number of data records present in given web pages
$DR_{correctly}$	–	number of correctly extracted data records
$DR_{extracted}$	–	number of extracted data records
$editDistance()$	–	function for calculating atree edit distance of a web page

E_W	–	a program that executes wrapper and extracts data
FN	–	false negative
FP	–	false positive
$max()$	–	a function returning the biggest number
ND	–	normalized string edit distance
$Nodes()$	–	number of nodes in given HTML tree
$NSTM$	–	normalized simple tree matching distance
p, q	–	integers
P_1, P_2	–	web pages
$P^{p,q}(T_i)$	–	pq-grams of given tree data structures
$pqGDist()$	–	pq-gram distance between given trees
r	–	resemblance of given text documents
$s()$	–	function generating shingle set from given document
s_1, s_2	–	strings
STM	–	simple trees' matching score
T_1, T_2	–	tree data structures
$TED()$	–	HTML tree edit distance
TN	–	true negative
TP	–	true positive
T_S	–	source data schema
T_w	–	target data schema
VW	–	visual weight of web page element
W	–	data extracting wrapper (rules set for data extraction)
$xp(P_i)$	–	a bag of XPath strings from a given web page

Abbreviations

AJAX	–	Asynchronous JavaScript and XML
CPU	–	Central Processing Unit
CSS	–	Cascading Style Sheets
DOM	–	Document Object Model
FTP	–	File Transfer Protocol
GB	–	Gigabyte
GHz	–	Gigahertz
HTML	–	Hypertext Markup Language

RAM	–	Random-Access Memory
RPM	–	Revolutions per minute
WWW	–	World Wide Web
XML	–	Extensible Markup Language
XPath	–	a query language for selecting nodes from an XML document
PHP	–	server-side scripting language designed for web development
PERL	–	general-purpose, interpreted, dynamic programming language
ASP	–	a server-side script engine for dynamic web pages

Contents

Introduction

The Investigated Problem

Structured data on the World Wide Web (*Web*) is usually embedded into template-generated web pages (Cafarella *et al.* 2011). Typically, upon a page request, this data is retrieved from databases and inserted into a web page using some fixed style templates. There are thousands of web pages on the Web that differ in visual style and underlying structure. Automatically extracting structured data from these template-generated web pages is not a trivial task and much of research efforts in the field of information extraction are put into tackling the problem (Bohannon *et al.* 2012; Dalvi *et al.* 2011; Furche *et al.* 2012b).

Traditional information extraction techniques considered in the database community tend to be aimed at extracting target information from manually specified sources (Bohannon *et al.* 2012). Although there are some proposals to extract data automatically (Crescenzi 2001; Liu *et al.* 2010; Zhai, Liu 2006; Zhao *et al.* 2005), but, unfortunately, these and many other state-of-the-art solutions do not demonstrate required level of precision and recall high enough to power real applications (Bohannon *et al.* 2012). Web-scale data extraction is remaining an open challenge (Blanco *et al.* 2011).

In this thesis we study automatic structured data extraction from template-generated web pages. Two methods are proposed that have potential to be implemented into Web-scale data extraction systems.

Importance of the Thesis

Data on the Web is about everything. An ability to automatically locate, extract and integrate that data can bring huge benefits (Madhavan *et al.* 2007). Indeed, it can fundamentally shift the search to allow a more semantic view of content (Dalvi *et al.* 2009). Currently search engines see the Web as hyper-linked pages, each containing a bag of words. A user searching the Web is presented with list of links to the web sites. However, the true value of the Web lies in the wealth of information provided by those pages (Dalvi *et al.* 2009). So instead of following the links, search engine users could instantly see the exact information they were looking for, such as list of products, movies, books, flights, real estate listings or any other real world entities (Weikum, Theobald 2010). Furthermore, Web-scale data extraction could enable and speed up data integration from multiple web sources. This would lead to building enormous size knowledge bases. A long standing goal for the Web search, that is – to return answers in the form of facts, would be much more tangible (Cafarella *et al.* 2011). So Web-scale structured data extraction could fundamentally change the way we see and browse the Web today.

Furthermore, automatic web data extraction solutions are also important for companies. The success of todays' company hinges on identifying and responding to competitive pressures (Connotate 2012). Many companies gather information from online sources and such activity belongs to online business intelligence (Baumgartner *et al.* 2009b). One of the key objectives of online business intelligence is to collect valuable information from many web sources to support decision making and thus gain competitive advantage. However, the online business intelligence presents non-trivial challenges to web data extraction systems that must deal with technologically sophisticated modern web pages where traditional manual programming approaches often fail to successfully extract data. Web sites are designed differently. If a company wants to collect data from hundreds of sources it is practically infeasible to try manually write data extracting rules for each source. A fully automatic web data extraction approaches should be used to achieve cost effective results. However, automatic web data extraction is still an active research area. Even if there are a limited number of web pages to be monitored a business intelligence system could have some degree of automation to reduce human efforts needed to build extraction rules and thus save costs to organization (Ferrara *et al.* 2012). In best case scenario the user

without any programming knowledge should be able to include easily a new web site to be monitored or fix an old one. Furthermore, online business intelligence systems extracting data from many sources would further benefit by automatically adapting data extraction rules to constant changes of web sites (Dalvi, Bohannon 2009).

As we see, fully-automatic structured data extraction techniques would help to improve todays' web search, enable companies to reduce costs and gain competitive advantage.

The Object of Research

The object of the study is structured data extraction from template-generated web pages.

The Goal of the Thesis

The main goal of the thesis is to propose a novel more effective method for extracting structured data from template-generated web pages.

The Tasks of the Thesis

In order to achieve the goal, the following tasks have to be solved:

1. To review state-of-the-art data extraction techniques.
2. To analyse technologically sophisticated modern web pages containing structured data.
3. To propose a method capable of extracting structured data from technologically sophisticated modern template-generated web pages;
4. To propose a method for clustering structurally similar template-generated web pages into clusters.
5. To experimentally evaluate the proposed methods and compare the results to other state-of-the-art techniques.

Research Methodology

To achieve the goal, the following research methods are employed:

1. Exploratory research method is used while studying the object of research and reviewing the state-of-the-art.

2. Constructive research method is employed to develop and test the proposed methods for extracting structured data and structurally clustering template-generated web pages. The proposed methods are implemented as prototypes using Perl, Python and JavaScript programming languages.

Importance of Scientific Novelty

The main scientific contributions of our research are the following:

1. A novel method called ClustVX is proposed for extracting structured data from template-generated web pages. The method is based on clustering visually and structurally similar web page elements and it can extract structured data from web pages of any design where more than one data record is present. The method can handle technologically sophisticated modern web pages. Experimental evaluation results reveal that the method achieves higher than 98% precision and recall.
2. The proposed ClustVX method is domain-independent as it does not require any *a priori* knowledge about target data. The method is also unsupervised as it does not require any learning or any manual human effort to induce wrappers and extract structured data.
3. A novel method called UXClust is proposed for structurally clustering template-generated web pages. The method is unsupervised and it exploits a novel idea to leverage XPath addresses of inbound inner-site links to radically speed up web page clustering time. Experimental evaluation results reveal that the method can cluster more than 1 million web pages in less than 4 minutes in turn achieving higher than 90% precision and recall.

Practical Significance of Achieved Results

The proposed novel method for structured web data extraction can be used to automatically extract structured data records from template-generated web pages. Since no learning or manual work is required the method could significantly reduce costs for companies which gather data from web pages. Furthermore, the method can generate XPath wrappers that can be reused in Web-scale data extraction systems.

The proposed novel method for structural web page clustering can be used to automatically cluster template-generated web pages according to their structural

similarity. Since the method has very low computational complexity it is applicable in Web-scale structured data extraction and web page clustering systems.

The Defended Statements

In what follows the defended statements of this thesis are presented:

1. Clusters with XPaths of visually and structurally similar web page elements can be leveraged to detect repeating patterns of embedded structured data records in template-generated web pages.
2. XPaths of inbound inner-site links can be leveraged to significantly speed up clustering time of template-generated web pages.

Approval of the Results

Research results related to the dissertation subject are published in 8 scientific publications. Four of them are published in reviewed scientific journals. Two of the journals are included in the Thomson Reuters Science Citation Index.

The author has also made 4 presentations at international scientific conferences, 2 presentations at international scientific workshops and 1 presentation at international summer school:

- 10th International Baltic Conference on Databases and Information Systems (Baltic DB&IS 2012). July 8–11, 2012, Vilnius, Lithuania;
- 12th International Conference on Web Engineering (ICWE 2012). July 23–27, 2012, Berlin, Germany;
- 18th International Conference on Information and Software Technologies (ICIST 2012). September 13–14, 2012, Kaunas, Lithuania;
- 6th ACM International Conference on Web Search and Data Mining (WSDM 2013). February 4–8, Rome, Italy;
- 3rd Workshop on Data Extraction and Object Search (DEOS 2013). July 6–8, 2013, Oxford, United Kingdom;
- 12th Estonian Summer School on Computer and Systems Science (ESSCaSS 2013). August 18–22, 2013, Voore, Estonia;
- 4th International Workshop on Data Analysis Methods for Software Systems. December 5–7, 2013, Druskininkai, Lithuania.

Dissertation Structure

The dissertation consists of introduction, four main chapters, general conclusions, references, a list of publications by the author on the topic of the dissertation and a summary in Lithuanian. The total scope of the dissertation is 123 pages, 23 equations, 37 figures and 20 tables.

Acknowledgements

First of all I would like to thank my supervisor Prof Antanas Čenys for his continuous support and guidance throughout my PhD studies. I truly appreciate his time and effort spent with me.

I am very grateful for Prof Olegas Vasilecas and Prof Gintautas Dzemyda for reviewing my dissertation and providing me with very helpful comments and suggestions.

I would like also to thank all my friends and colleagues, especially Dr Lukas Radvilavičius, Dr Juozas Gordevičius and Assoc Prof Dr Nikolaj Goranin for countless invaluable discussions, comments and suggestions. Furthermore, without proper and very timely encouragement from Dr L. Radvilavičius I would hardly have started my PhD studies at all, thank you Dr Lukas!

I also especially thank my friend Aušrinė Benediktavičiūtė for her support and encouragements during my PhD studies and for proof-reading my dissertation.

I am also deeply thankful my family and will be eternally indebted to my parents who brought me into this wonderful world where I can pursue my dreams and live in peace.

Finally I thank all the people who somehow helped or supported me during my PhD studies.

1

Structured Web Data Extraction, Methods and Applications

In this Chapter we introduce the structured web data extraction research problem, review state-of-the-art methods and discuss real life applications for structured web data extraction systems.

Web data extraction is the problem of extracting target information from web pages. There are two general problems: extracting information from natural language text and extracting structured data from web pages (Bing 2012). The key difference in web data extraction is that a large part of the data on the Web is structured (Cafarella *et al.* 2011) by Hyper Text Markup Language (HTML) and visual styling, especially when web pages are automatically generated and populated from templates and underlying databases. This sets web data extraction apart from information extraction where entities, relations, and other information are extracted from free text, which, of course, may also come from web pages (Furche *et al.* 2012a).

Extracting information from free text is studied mainly in the natural language processing community (Bing 2012), while this review concentrates on extracting structured data from web pages.

Parts of this Chapter are published in (Grigalis, Čenys 2014a), (Grigalis, Čenys 2014b), (Grigalis, Čenys 2013), (Grigalis 2013), (Grigalis 2012), (Grigalis *et al.* 2012b).

1.1. Preliminaries

In this Section we review the main concepts, background techniques and challenges facing structured data extraction from modern web pages.

1.1.1. Structured Web Data

Generally, structured data on the Web (web data) are data records retrieved from underlying databases and displayed in web pages using some fixed templates (Bing 2012; Cafarella *et al.* 2011; Dalvi *et al.* 2012). In such database-backed web sites, each time a visitor request a web page the server-side scripting language engine, such as PHP, PERL, ASP or any other, issues a query to an underlying database, retrieves structured data records and embeds them into HTML template. Such web pages may also be referred to as template-generated.

Structured data in web pages can be presented in many visual formats, including HTML tables, horizontal lists, vertical lists, and etc. Each such page and their collections can be seen as online datasets (databases). For an example consider Figure 1.1 where a snippet from database-backed online shop's web page is shown. The snippet contains three data records, i.e. three products (boots).

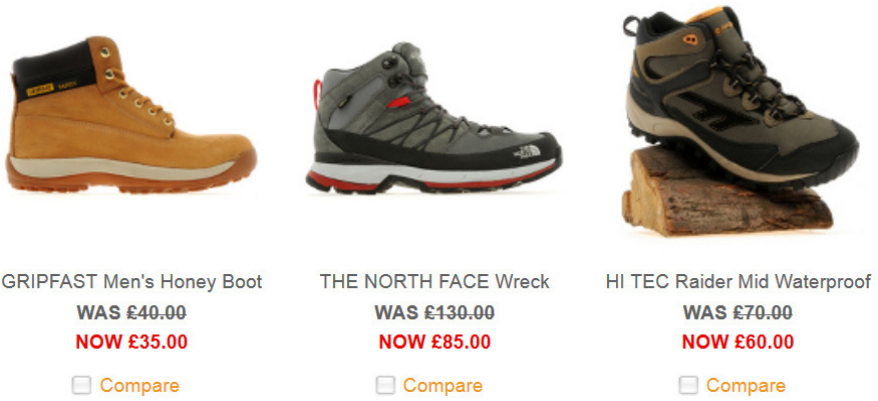


Fig. 1.1. An example of three data records rendered as a list

According to (Cafarella *et al.* 2011) structured data on web pages differs from data stored in traditional relational databases in several ways. First, data must be extracted from "page context". To the human user the data records (as seen in Figure 1.1) appears to be clearly structured, as title, price, picture and other information can be easily identified and seen as placed in an invisible table. However, a computer program seeing just the HTML source code must be able to

automatically distinguish valuable information from, say, HTML markup tags. So, according to (Cafarella *et al.* 2011), there is nothing akin to traditional relational metadata that leaves no doubt as to how many tables there are and the relevant schema information for each table.

Second, there exist no centralized data design or data quality control. In a traditional database, the relational schema provides a topic-specific design that must be observed by all data elements. The database and the schema may also enforce certain quality controls (such as observing type consistency within a column, disallowing empty cells, and constraining data values to a certain legal range) (Cafarella *et al.* 2011).

Third, there is vast number of topics. A traditional database typically focuses on a particular domain (such as products or proteins) and therefore can be modelled in a coherent schema. On the Web, data covers everything, and is also one of its appeals. The breadth and cultural variations of data on the Web make it inconceivable that any manual effort would be able to create a clean model of all of it (Cafarella *et al.* 2011).

(Bing 2012) observes that typically there are two types of structured data rich pages:

1. *List pages*: Each of such pages contains several lists of data objects. Figure 1.1 shows a snippet of such a page, which has a list three of products. From a layout point of view, we see that each data record is formatted using the same style, i.e. the same template. So list pages contain more than one data record formatted using some fixed style.
2. *Detail pages*: Such a page focuses on a single object and typically contains one data record. For example, in Figure 1.2, the page focuses on the product "Men's Honey Boot Twister Industrial Shoes". That is, it contains all the details of the product, including title, image, manufacturer, price and other purchasing information, product overview, available sizes, etc.

1.1.2. Definition of the Web Data Extraction Problem

As we already know, structured data found in template-generated web pages typically come from underlying relational database and upon a page request is embedding into a web page using some fixed HTML templates. Data extraction may be viewed simply as the reverse engineering task. That is, given the HTML mark-up encoded data (i.e., web pages), the extraction process recovers the original data model and extracts data from the encoded data records (Bing 2012).

Structured data extraction process can also be seen as a wrapper construction task (Doan *et al.* 2013). A wrapper is typically a set of rules, often presented as a computer program that can be repeatedly re-used to extract structured data.

Fig. 1.2. An example of detail page with embedded structured data record

Given a source *S,* such as a web page with embedded structured data, a wrapper *W* extracts that data from the source *S*. Formally, *W* is a tuple (T_W, E_W), where T_W is a *target schema*, and E_W is structured data *extraction program* that uses the format F_S to extract from each page a data instance conforming to T_W. The target schema T_W need not be the same as the schema used on the page, since we may want to give another name to the attributes or only extract a subset of them (Doan *et al.* 2013). Please see the following two examples found in (Doan *et al.* 2013) which illustrate the operation of wrappers.

As a first example, consider a wrapper *W* that extracts all attributes from the data records shown in Figure 1.1. The target schema T_W is the source schema T_S = (*image, title, price_last, price_now*). The extraction program E_W may run in a way that, when given a page *P* from the source, extracts the first image as product image, the text below the image as a product title, then the string immediately following "WAS" and formatted as strike-trough text as last price, and so on.

As a second example, consider a wrapper *W* that extracts only title and current price, see Figure 1.1. Here the target schema T_W (*title, current_price*) is a subset of source schema T_S (*image, title. price_last, price_now*), and the extraction program E_W extract only the title and current price.

So the *wrapper construction* problem is to quickly create the pair (T_W, E_W) by inspecting the pages of the source *S.* We want to learn the source schema T_S as well as a program E_W that extracts data conforming to T_S. Thus, formally, the wrapper to be constructed is shown in Formula 1.1 (Doan *et al.* 2013):

$$W = (T_S, E_W). \quad (1.1)$$

In the need to extract multi-attribute data records a method automatically constructs a wrapper W which could extract a target schema T_W. The target schema should contain all attributes found in the source schema T_S. Considering the three data records (input) shown in Figure 1.1, the desired extraction results (output) are shown in the Table 1.1.

Table 1.1. Desired extraction results

Image1	GRIPFAST Men's Honey Boot	WAS £40.00	NOW £35.00	Compare
Image2	THE NORTH FACE Wreck	WAS £130.00	NOW £85.00	Compare
Image2	HI TEC Raider Mid Waterproof	WAS £70.00	NOW £60.00	Compare

As seen in Table 1.1, the extracted text strings "*WAS*", "*NOW*", "*£*" and "*Compare*" may be absent in the original product schema found in relational database table, however, without semantic analysis and/or entity resolution a wrapper cannot easily tell if those strings belong to the fixed HTML template or product schema. Thus a wrapper usually aims to extract all found structured data with all attributes and leave the data (schema) cleaning problem as a separate step.

1.1.3. HTML Document Object Model and XPath

The Document Object Model (DOM) is a "platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents"[1]. The DOM is standardized as a W3C recommendation[2].

Following the DOM, a web page is represented as an ordered tree structure which is a fundamental data structure in XML documents. See Figure 1.3 for an example. Thus a very useful approach for extracting structured data from Hypertext Markup Language (HTML) documents is to employ Extensible Markup Language (XML) technologies to translate HTML to valid XML code. In this approach, HTML documents are first normalized into Extensible HTML (XHMTL) and then then processed by XML applications (Myllymaki, Jackson 2002).

XPath[3] is a language used to navigated XML tree structure. The primary purpose of XPath is to access parts of an XML document. XPath is the result of

[1] http://www.w3.org
[2] http://www.w3.org/DOM
[3] http://www.w3.org/TR/xpath20

an initiative to create a common syntax and semantics for functionality sharing between XSL Transformations and XPointer. XPath operates on the abstract, logical structure of an XML document, rather than its underlying syntax (source code). XPath gets its name from its use of a path notation as in URLs for navigating through the hierarchical structure of an XML document. XPath sees an XML document as a tree of nodes (see Figure 1.3). The modelled HTML tree is widely used in structured data extraction algorithms (Bohannon *et al.* 2012; Chang, Kuo 2004; Chang 2001; Dalvi, Bohannon 2009; Etzioni *et al.* 2011; Myllymaki, Jackson 2002), where very commonly dynamic programming is used to find similar branches of the tree. Similarity is determined by comparing every tree nodes with each another and if calculated similarity is higher than a predetermined threshold, then two branches of the tree are similar.

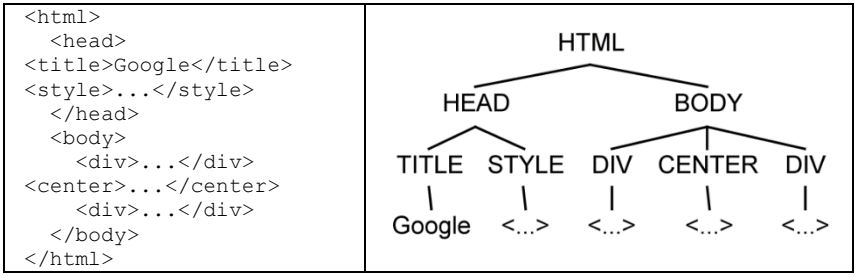


Fig. 1.3. HTML source code on left represented as a tree structure on right

A location path (XPath) is one of most important kind of expressions in XPath language. An XPath locates and selects a set of nodes relative to the context node (usually a root node of the tree). The result of evaluating an XPath is the node-set containing the nodes selected by the XPath. Every XPath can be expressed using a straightforward but rather verbose syntax. There are also a number of syntactic abbreviations that allow common cases to be expressed concisely (Clark *et al.* 1999).

There are two kinds of XPath: relative location XPath and absolute location XPath. A relative XPath consists of a sequence of one or more location steps separated by /. The steps in a relative location path are composed together from left to right. Each step in turn selects a set of nodes relative to a context node. Relative XPath can improve the robustness of extraction wrapper (Dalvi, Bohannon 2009). An absolute XPath consists of / optionally followed by a relative location path. A / by itself selects the root node of the document containing the context node. If it is followed by a relative location path, then the location path selects the set of nodes that would be selected by the relative location path relative to the root node of the document containing the context node (Clark *et al.* 1999).

For an example, consider the two kinds of XPath expressions in Figure 1.4. Both evaluated on HTML tree presented in Figure 1.3 would return the same results: text node with string "*Google*".

Absolute XPath →	/html/head/title
Relative XPath →	//title

Fig. 1.4. An example of relative and absolute XPath expressions

1.1.4. Challenges Posed by Modern Web Pages

HTML originally was meant only to convey the logical structure of a document and not its visual rendering. A web browser was free to visually display the page uniquely to its own specifications, as long as the overall structure of the HTML document was preserved. So it was more important to convey the content rather than the form (Berners-Lee 2000; Thomsen 2013).

Web is evolving. Today, the focus is being placed on precise visual rendering, rather than conveying the internal structure of the document. This change of perspective means that just looking at the textual source code of a web page does not necessarily reveal the structure of the document and hence the content structure (Thomsen 2013).

Moreover, to enhance browsing experience modern web pages rely on a number of sophisticated technologies, such as Cascading Style Sheets (CSS) to separate presentation style of web pages from their content, Asynchronous JavaScript Requests (AJAX) to dynamically load web page content, client side scripting to modify the appearance of a web page solely on client side and etc. Modern web browsers are complicated software that requires considerable amount of computational power to visually render web pages. Looking from a user perspective, all these technological advances of the Web enhance browsing experience. However, on the other hand they hinder web data extraction.

Having the HTML document focus on form rather than content means that it is non-trivial for computer programs to extract information from them (Thomsen 2013). To extract embedded structured data is not enough to simply download a web page from a server and analyse its source code. Modern web pages are sometimes loaded incrementally using AJAX requests. JavaScript can modify the asynchronously received data according to some algorithms and only then display it on a page. For example, some modern web sites use AJAX technique for pagination: when user clicks "next page" link, a browser does not reload all page, instead – only a required data is asynchronously retrieved from a web server and displayed as a next page. Even more, each step of navigation inside a web site can

be controlled by cookies and referrers. All these and many other features of a modern web site for data extraction means only one thing – each data extracting tool should fully emulate a modern web browser. It is an incredibly hard task.

1.2. Structured Web Data Extraction Techniques

Structured web data extraction approaches can be classified according to the main technique used by each approach to generate a wrapper. (Laender *et al.* 2002a) suggest using the following classification:

1. Languages for Wrapper Development.
2. HTML-aware Tools.
3. NLP-based Tools.
4. Wrapper Induction Tools.
5. Modelling-based Tools.
6. Ontology-based Tools.

Of course, while such taxonomy is useful for didactic purposes, it must not be taken as the only possibility. In fact, there are cases where a same method could fit in two or more of the identified groups. However, the proposed taxonomy is helpful as a guide for properly understanding the existing approaches to web data extraction (Laender *et al.* 2002a).

In what follows, a description is provided for the main characteristics of the methods belonging to each group.

Languages for Wrapper Development are one of the first proposals for addressing the problem of wrapper generation. Some of the best known tools that employ this approach are Minerva (Califf, Mooney 1999), TSIMMIS (Hammer *et al.* 1997), web-OQL (Arocena, Mendelzon 1998), FLORID (Laender *et al.* 2000), and Jedi (Huck *et al.* 1998). These techniques encompass the development of languages specially designed to assist users in constructing wrappers. Such languages were proposed as alternatives to general purpose languages such as Perl, Java, Python and etc., which were prevalent so far for wrapper coding task (Laender *et al.* 2002a).

HTML-aware Tools rely on inherent structural features of HTML documents for accomplishing data extraction and wrapper generation tasks. Some representative tools based on such an approach are W4F (Sahuguet, Azavant 2001), XWRAP (Ling *et al.* 2000), RoadRunner (Crescenzi 2001), DEPTA (Zhai, Liu 2006) and many other that are covered later in this section. The techniques covered by this group typically work on HTML tree structure, i.e. a representation that reflects its HTML tag hierarchy, and search for repeating patterns. Wrappers are generated either semi-automatically or automatically (Laender *et al.* 2002a).

NLP based Tools. Natural language processing (NLP) techniques typically employ various natural language processing techniques, such as filtering, part-of-speech tagging, and lexical semantic tagging to build relationship between phrases and sentences elements, and finally extract facts from free text. A fact is represented as a tuple consisting of entities (real world objects) and their relationship (Laender *et al.* 2002a). For instance, given the sentence, "McCain fought hard against Obama, but finally lost the election," a data extraction system of this kind should extract two tuples, (*McCain, fought against, Obama*), and (*McCain, lost, the election*) (Etzioni *et al.* 2011). This way millions of facts are extracted from Web-scale text corpus and put into knowledge bases. Representative tools based on such an approach are RAPIER (Califf, Mooney 1999), SRV (Freitag 2000), WHISK (Soderland 1999), (Etzioni *et al.* 2011; Suchanek *et al.* 2007).

Wrapper Induction Tools. The wrapper induction tools generate delimiter-based extraction rules derived from a given set of training examples. They rely on formatting features that implicitly delineate the structure of the pieces of data found. This makes such tools more suitable for HTML documents than the previous ones. Tools such as WIEN (Kushmerick 1997), SoftMealy (Britain *et al.* 1998), STALKER (Muslea *et al.* 2001), IEPAD (Chang 2001), Lixto (Baumgartner, Flesca 2001), are representative of this approach (Laender *et al.* 2002a).

Modelling-based Tools, given a target structure for objects of interest, try to find in web pages portions of data that implicitly conform to that structure. The structure is provided according to a set of modelling primitives (e.g., tuples, lists, etc.) that conform to an underling data model. . Tools that adopt this approach are NoDoSE (Adelberg 1998) and DEByE (Laender *et al.* 2002b). Following, algorithms similar to those used by the wrapper induction tools, the Modelling-based tools identify objects with the given structure in the target web pages (Laender *et al.* 2002a).

Ontology-based Tools. All previously described approaches rely on the structure of presentation features of the data within a web page to generate rules or patterns to perform data extraction. However, extraction can be also accomplished by relying directly on the data itself. Given a specific topic domain, ontology can be used to identify constants present in the page and to construct objects with them. (Laender *et al.* 2002a). The most prominent techniques belonging to this category are tool developed by the Brigham Young University Data Extraction Group (Embley *et al.* 1999), ODE (Su, Wang 2009), the system DIADEM (Furche *et al.* 2012b) developed by Oxford university.

Another important aspect by which data extraction tools can be classified is the level of required human input, such as labelling positive/negative example, ontology preparation, and etc. So, according to (Bing 2012), all data extracting

techniques, depending on the level of automation, can be classified into these three main categories:

1. Manual approaches.
2. Semi-automatic (supervised learning) approaches.
3. Fully Automatic (unsupervised) approaches.

In Manual approaches the human programmer, by analysing a web page and its source code, identifies some patterns and then writes a program to extract the target data. To simplify the process for programmers, several pattern specification languages and user interfaces have been built. However, the manual wrapper building approach is not scalable to a large number of sites (Bing 2012).

(Crescenzi *et al.* 2013) propose to leverage the power of the crowd to overcome inherited scalability limitation in data extracting systems where wrappers are generated from manually labelled examples. Crowd sourcing platforms, such as Amazon Mechanical Turk[4], present an opportunity to make the manual annotation process more affordable, also at large scale. So Crescenzi et al. introduce "a framework to support a wrapper inference system supervised by the crowd". Their framework aims at catching the opportunities of crowd sourcing, i.e. reducing wrapper creation costs and scaling the overall work.

Similarly, Google runs experimental Fusion Tables project[5] which encourages creating, improving and sharing data presented tables. This approach too tries to exploit the power of the crowd in a way that many users collaboratively collect and share data from many web sources (Gonzalez *et al.* 2010).

Supervised learning approaches usually build data extraction wrappers, which, in other words, are programs or strictly defined logical rules used to extract data from web pages. Supervised learning approaches need manually labelled sample pages to learn its rules (Baumgartner, Flesca 2001; Britain *et al.* 1998; Chang, Kuo 2004; Chang 2001; Kushmerick 1997; Laender *et al.* 2002b). The main two disadvantages of supervised learning approaches are time consuming manual labelling process and matching already inducted wrappers to constant change of web sites (Gulhane *et al.* 2011; Kushmerick 1997; Raposo *et al.* 2007). Even one web site may have many different web page templates. To manually label all of these templates a lot of human input is required. So the supervised learning approaches are usually not applicable to Web-scale structured data extraction and are used to extract data from several selected web sites of high interest.

The obvious advantage of automatic data extraction techniques (Álvarez *et al.* 2008; Crescenzi 2001; Hong *et al.* 2010; Jindal, Bing 2010; Kayed, Chang 2010; Liu 2005; Liu *et al.* 2003, 2010; Simon 2005; Su *et al.* 2011; Zhai, Liu 2006;

[4] http://www.mturk.com

[5] http://tables.googlelabs.com/

Zhao *et al.* 2005) is that they are able to extract structured data from different web pages without human intervention.

It is important to note, that much of current research effort (Bohannon *et al.* 2012; Dalvi *et al.* 2011; Elmeleegy *et al.* 2011; Furche *et al.* 2012b; Gulhane *et al.* 2011) is put into developing fully-automatic structured web data extraction techniques that could be applicable to extract data at Web-scale. Thus, in what follows, we in details review only automatic structured web data extraction techniques, which we classify into three categories: pattern search based, visual signals aided, ontology based. Readers more interested in manual or semi-automatic approaches are guided to read some comprehensive surveys by (Chang *et al.* 2006; Ferrara *et al.* 2012; Laender *et al.* 2002a; Lam, Gong 2005; Sleiman, Corchuelo 2013).

1.2.1. Pattern Search Based Methods

The key step in structured data extraction is to find the encoding template from a collection of encoded instances of the same type, i.e. we need to identify the repeated HTML tag patterns which encode data records. String matching and tree matching are obvious techniques for the task. Tree matching is useful because HTML encoding strings also form nested structures due to their nested HTML tags. Such nested structures, as we already know, can be modelled as trees, commonly known as HTML DOM trees (Bing 2012).

String edit distance (also known as Levenshtein distance) is one of the most widely used string matching/comparison technique (Bing 2012). String edit distance or its variation as HTML tag sequences comparison technique is used in MDR (Liu *et al.* 2003), ViPER (Simon 2005), PADE (Su, Wang 2009), VINTS (Zhao *et al.* 2005), (Álvarez *et al.* 2008). The edit distance of two strings is defined as the minimum number of character modifications required to change one string into another. Allowed modifications of any single character include: change a character, insert a character, and delete a character. Each modification has its own cost. The normalized edit distance *ND* of two strings s_1 and s_2 is defined as the edit distance (d) divided by the mean length of the two strings (Bing 2012):

$$ND(s_1, s_2) = \frac{d(s_1,s_2)}{(|s_1|+|s_2|)/2}. \quad (1.2)$$

According to (Bing 2012), another option for the denominator is to use $max(|s_1|,|s_2|)$, i.e. the length of the longest string.

Another widely adopted technique to automatically detect and extract data records is to search for repetitive patterns in HTML source code by calculating the similarity of HTML trees.

Similarly like string edit distance, the tree edit distance between two trees A and B (labelled ordered rooted trees) is the cost associated with the minimum set of modifications needed to transform A into B. Typically the set of allowed operations used to define tree edit distance includes: tree node removal, tree node insertion, and tree node replacement. Here again, a cost is assigned to each operation. Thus solving the tree edit distance problem is to find a minimum-cost mapping between trees A and B. During the comparison each node should appear no more than once in a mapping and the order among siblings and the hierarchical relationships should be preserved (Bing 2012). Figure 1.5 shows a simple tree mapping example.

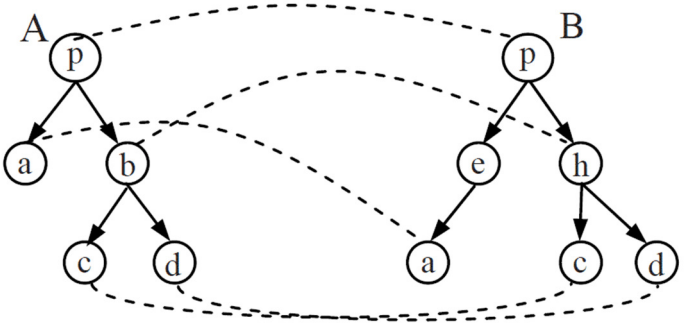


Fig. 1.5. A general tree mapping example (Bing 2012)

In practice, a variation called simple tree matching (STM) algorithm (Yang 1991) is employed in various data extraction methods to find similar sub trees in HTML document. It is a restricted tree mapping variation in which no node replacement and no level crossing in hierarchical relationships are allowed. In STM, the aim is to find the maximum matching between two trees and not the edit distance. The normalized simple tree matching distance *NSTM* of the two trees *A* and *B* can be obtained by dividing the trees matching score *STM* by the mean number of *nodes()* in the two trees (Bing 2012):

$$NSTM(A,B)\frac{STM(A,B)}{(nodes(A)+nodes(B))/2}. \tag{1.3}$$

Here again, the denominator can be replaced with *max(nodes(A), nodes(B))*, where *nodes(X)* is the number of nodes in tree *X* (Bing 2012).

(Zhai, Liu 2006) were one of the first to exploit simple tree matching algorithm for many automatic structured data extraction systems called DEPTA (Zhai, Liu 2006), NET (Liu 2005), and G-STM (Jindal, Bing 2010). The latest state-of-the-art G-STM system integrates the grammar based approach and tree matching to produce a brand new algorithm, which is a more principled approach to extract structured web data from a web page (Jindal, Bing 2010). G-STM generalizes a tree matching algorithm (Zhang, Shasha 1989) and introduces special grammar generation method so that it can identify and deal with lists inside data records.

Some other automatic structured data extraction systems called FiVaTech (Kayed, Chang 2010), CTVS (Su *et al.* 2011) (and its predecessor DeLa (Wang, Lochovsky 2003)), WISH (Hong *et al.* 2010) work very similarly to G-STM and depend on the same tree matching technique (Yang 1991).

M. Alvarez et al. propose another structured data extraction method (Álvarez *et al.* 2008) which is not based on simple tree matching like all above systems. Instead of comparing nodes in HTML tree directly, the authors first convert them to a string and then calculate string edit distance to determine the similarity. Their method constitutes three main steps. First, the method begins by finding the dominant data region in a web page. Then, it performs a clustering process to limit the number of candidate record divisions in the dominant data region. Each candidate record list will propose a particular division of the data region into records. After that, their system chooses the one having highest similarity according to string edit distance calculations. The similarity between records in a data region is determined by comparing two sequences of consecutive sibling sub trees in the HTML tree of a page in each of the record. Finally, a multiple string alignment algorithm is used to extract the attribute values of each data record. This means that after selecting an initial data record as a starting string, each additionally selected data record is aligned to the so far obtained string, until all data records are aligned to the string. The same data items alignment technique is also used in DEPTA (Zhai, Liu 2006) system.

(Miao *et al.* 2009) introduce a tag paths clustering concept. Their method focuses on how a distinct tag path appears repeatedly in the DOM tree of the web page. Instead of comparing a pair of individual segments, it compares a pair of tag path occurrence patterns (called visual signals) to estimate how likely these two tag paths represent the same list of data records. However, the method does not actually retrieve the rendered visual information about each tag path. The concept of visual signals is based on one key assumption, that each tag path defines a unique visual signal. Furthermore, the method just segments a web page region into data records and does not extract information that is inside data records.

1.2.2. Visual Signals Aided Methods

Contrary to the above systems, which search for repeated patterns purely in HTML code, VINTS (Zhao *et al.* 2005) system utilizes both visual content features and HTML tree structure regularities of web page to detect structured data. This system also employs a clustering technique: to identify the main data region containing data records VINTS first identifies content line separators (visual feature) and uses them to segment result page into visual block. See Figure 1.6 where an example of a data record (a), its block shape (b) and other possible shapes (c) are present.

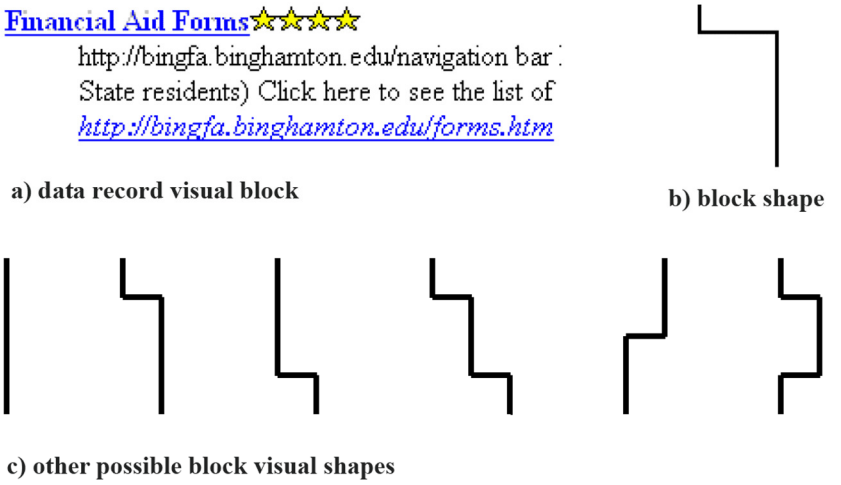


Fig. 1.6. An example of visual signals (block shapes) that are used in VINTS system (Zhao *et al.* 2005)

Then the blocks that are consecutive and visually similar are clustered into one group. The clustering is based on visual similarity, which means that two blocks are visually similar if their type distance, shape distance and position distance are all below certain hardcoded thresholds. After that, VINTS uses heuristics to determine which of blocks contain data records.

A more recent automatic structured data extraction system VIDE (Liu *et al.* 2010) tries not to depend on HTML tree at all and instead uses purely visual features of a web page. Using patented VIPS (Cai *et al.* 2003) algorithm it builds a visual containment tree of a web page and uses it instead of pure HTML tree. However if there are some unloaded images or missing style information in a web page VIPS may fail to build correct visual containment tree which leads to data extraction problems (Liu *et al.* 2010).

Some authors (Furche *et al.* 2011; Walther 2012) have also addressed the problem of rendering modern web pages in a browser, accessing visual signals, and only then extracting data. To render a web page they employ frameworks originally used for web application testing (HTMLUnit[6], Selenium[7], FireWatir[8], and etc.). Such frameworks control modern web browsers, such as Mozilla Firefox, Chrome, or Internet Explorer.

1.2.3. Ontology Based Methods

One of the most promising advances is DIADEM project[9] at Oxford University. The acronym stands for Domain-centric, Intelligent, and Automated Data Extraction Methodology (Furche *et al.* 2012b). The project is fundamentally different from many previous approaches to automatically extract web data at large scale. The integration of state-of-the-art technology with reasoning using high-level expert knowledge at the scale envisaged by DIADEM team has not yet been attempted and has a chance to become the leading example of next generation web data extraction systems. Without human supervision the DIADEM system locates, navigates, and analyses web sites of a specific domain (such as cars, books, products, real estate, flights and etc.) and extracts all contained structured data objects using highly efficient, scalable, automatically generated wrappers. The web page analysis is parameterized using domain knowledge that allows DIADEM to replace human annotators and to refine and verify the generated wrappers. This approach works in contrast to other modern web data extraction systems which require human annotators to manually mark data on target web pages and to record web site navigation. However, there seems to be two remaining challenges: simplifying the process of domain knowledge creation and adding the support for other languages. Currently DIADEM works only with web sites in English language. We see it as a limitation, since there may be many customers interested in extracting data from non-English web sites.

(Bohannon *et al.* 2012) from Yahoo develop a system tuned for automatic Web-scale information extraction. Like in DIADEM project the system is *domain-centric*. This means that there is a human supervision at domain level, i.e. humans should define some rules about the topic of interest. The authors present an example that if we are interested in constructing a database of restaurants from the Web, we can specify the set of attributes that we are interested in, e.g. “name”, “address” and “reviews”, supply sample dictionaries or regular expressions or

[6] http://htmlunit.sourceforge.net
[7] http://docs.seleniumhq.org
[8] http://watir.com
[9] http://diadem.cs.ox.ac.uk/

language models for attributes, specify domain knowledge like “businesses typically have a single phone number but multiple reviews”, and so on. Bohannon et al. believe that solving the domain-centric extraction can provide a promising stepping stone towards cracking the grand challenge of a general Web-scale information extraction.

1.3. Web Page Clustering for Data Extraction Techniques

Template-generated web pages contain most of structured data on the Web. Clustering these pages according to their template structure is an important problem in wrapper-based structured data extraction systems. These systems extract structured data using wrappers that must be matched to only particular template pages. Selecting single type of template from all crawled web pages is a time consuming task. Conceptually, each cluster corresponds to the output of one of the template generating scripts that created the site. Alternatively, if manual work is done to select which pages to wrap, the benefit of unsupervised extraction techniques is effectively lost, since non-trivial human effort requiring work must still be done per site (Blanco *et al.* 2011).

Thus in this Section we review web page structural clustering methods which automatically can organize web pages into clusters according to their structural similarity.

1.3.1. Cross-linkage Based Methods

The idea that linkage among web pages can reveal something about their similarity is not new. For example, Small (Small 1973) claims that the relationship between two documents can be calculated by analyzing how often they both are cited in the same source. Dean et al. (Dean, Henzinger 1999) and Spertus (Spertus 1997) brings the same idea to the context of web pages. Both works present methods to detect topically related pages without actually downloading and analysing their content. Instead, proposed algorithms analyse web pages for co-linkage and links placement. For example, densely placed links (Spertus 1997) or links having many same parent nodes (Dean, Henzinger 1999) in HTML tree are considered to be linking to topically similar web pages.

Crescenzi et al. (Crescenzi *et al.* 2005) further demonstrate that the analysis of co-linkage and link placement holds true and for structural similarity of web pages. They propose an algorithm that crawls given web site and incrementally builds site model for it. Their work is closely related to focused-crawling (Chakrabarti *et al.* 1999; Diligenti *et al.* 2000) research field where focused-

crawling systems are usually built to crawl topically related web pages, however, Crescenzi et al. look for structural similarity. The structural similarity is also determined by comparing the placements of link collections. In other words, two pages are considered to be structurally similar if they both have groups of links placed in the same locations. One of the main observations they present and validate is that links sharing the same layout and presentation properties usually point to pages that are structurally similar. Same idea is also employed by Lin et al. (Lin *et al.* 2010) to hierarchically cluster web pages.

1.3.2. Text Content Based Methods

Other early works look for a way to identify duplicate web content. This is very important problem in information retrieval field and more specifically to search engines were it is crucial to return most relevant and unique content. Links to duplicated content in most cases have no value for the user and search engines do their best to avoid presenting duplicated content. To deal with this problem (Broder *et al.* 1997) proposed to calculate resemblance and containment of two web pages. The resemblance of two web pages is described as they are "roughly the same". Similarly, the containment of two web pages indicates that one is "roughly contained" within another. Each web page document *D* is viewed as a sequence of words. This sequence is divided into tokens called *shingles*. Each shingle can contain a chosen *w* amount of characters. Then sets of shingles from two web page documents can be compared to determine their resemblance and containment. Formally, given a document D we define its *w-shingling S(D, w)* as the set of all unique shingles of size *w* contained in *D*. So for example, the *4-shingling* of text content (Broder *et al.* 1997):

$$(a, rose, is, a, rose, is, a, rose), \tag{1.4}$$

results into a shingle set:

$$\{ (a, rose, is, a), (rose, is, a, rose), (is, a, rose, is) \}. \tag{1.5}$$

So for a given shingle size, the resemblance *r* of two documents D_1 and D_2 is defined as:

$$r(D_1, D_2) = \frac{|S(D_1) \cap S(D_2)|}{|S(D_1) \cup S(D_2)|}, \tag{1.6}$$

where $|s(D_n)|$ is the size of a shingles set from document D_n.

The containment c of document D_1 in D_2 is defined as:

$$c(D_1, D_2) = \frac{|S(D_1) \cap S(D_2)|}{|S(D_1)|}. \quad (1.7)$$

Hence the resemblance is a number between 0 and 1, and it is always true that $r(D_1, D_1) = 1$, i.e. that a document resembles itself 100%. Similarly, the containment is a number between 0 and 1 and if D_1 is fully contained in D_2 then $c(D_1, D_2) = 1$ (Broder *et al.* 1997).

To scale up this approach and avoid computationally expensive shingle generation and comparison (Broder *et al.* 1997) proved that it suffices to keep for each web page a sketch of a few hundred bytes. The sketches can be efficiently computed (in time linear in the size of the documents) and, given two sketches, the resemblance or the containment of the corresponding documents can be computed in time linear in the size of the sketches.

To simplify and speed up structural comparison of web pages Buttler (Buttler 2004) proposed to adapt the shingles based content similarity detection technique (Broder *et al.* 1997) to compare structural content of web pages. In this adapted technique web pages are viewed as a set of sequence of branches, paths from the HTML tree root node to a leaf node. In such way HTML tree can be encoded as a list of these tokens. The proposed path similarity measure finds the similarity of paths between two different web pages. Path similarity can be calculated in several ways: binary, where a path is either equivalent or not; partial, where the number of comparable nodes in each path are matched; or weighted, where the nodes are weighted according to their distance from the root and longer paths have bigger influence on similarity. Following the shingles approach and applying it to the path structure, for each node in the tree representation of given web Page, Buttler (Buttler 2004) computes the path from the root to that node. Then a hash is created based on the list of tag names in that partial path. The derived sets of hashed values are used to compare two web Pages. Furthermore, to speed up similarity computation each page can get its fingerprint, i.e. a set of results from applying hashing functions to each shingle. This way a document is “sketched” into a few hundred bytes. The sketches can be efficiently computed (in time linear in the size of the documents) and, given two sketches, the resemblance or the containment of the corresponding documents can be computed in time linear in the size of the sketches [15].

1.3.3. HTML Tree Based Methods

Another early research field, related to web document similarity detection, concentrates on XML (Extensible Markup Language) document and their schema comparison (Blanco *et al.* 2011). One widely adopted method, called tree edit distance, measures the minimum number of node insertions, deletions, and updates required to convert one XML document tree into another. This can be converted into a similarity metric by normalizing the number of edit operations with the number of nodes in the tree representing the larger document (Buttler 2004). However, the computational complexity of techniques involving tree edit distance calculation (Demaine, Mozes 2007; Nierman, Jagadish 2002; Tai 1979; Zhang, Shasha 1989) is at least quadratic. To overcome this limitation and reduce required running time Augsten et al. (Augsten *et al.* 2005) propose approximating tree edit distance calculation technique called pg-grams. This technique, a reminiscent of web page shingling method (Broder *et al.* 1997), splits XML tree into smaller trees called pq-grams, where p and q describes the depth and width of the smaller trees. This way two XML trees are similar if they share many identical pg-grams. Augsten et al. (Augsten *et al.* 2010, 2005) claims that for two trees of size n, the pq-gram distance can be computed in O(n log n) time and O(n) space.

Tag similarity is another metric for determining structural similarity of two XML documents. This approach measures how closely the set of tags match between two documents. While these techniques are also applicable for structural clustering HTML pages, HTML pages are more difficult to cluster than XML documents because they are noisier, do not validate to simple/clean schemas, and are very homogeneous because of the fixed set of tags used in HTML (Blanco *et al.* 2011).

Joshi et al. (Joshi *et al.* 2003) proposed an alternative scheme for representing the structural information of web documents based on the paths contained in the corresponding HTML tree model. Since proposed model includes partial information about parents, children and siblings, it allows to derive meaningful and at the same time computationally simple structural similarity measure. The model has two variants. In the first one a bag of unique HTML tree tag paths without sibling information is used. In the second – a bag of all XPaths is used. The latter incorporates not only parent/child relationships but also sibling relationships in HTML tree. This way bag of XPaths model is more robust.

Later Chakrabarti et al. (Chakrabarti, Mehta 2010) proved that in most cases it is not necessary to compare all XPaths of two documents to determine their structural similarity. Instead, they observed that each template style in a web site has unique and important sections. For example, product page template can have product title displayed on same particular location in instance of that template. This way some HTML tree locations are more important than other and can aid in

discerning one template from another. Authors name these important locations in HTML tree as “key” paths. The web pages on the web site are then clustered using these “key” paths. To automatically identify these “key” paths authors suggested using search engine logs: proposed algorithm requires information on search queries, and the web pages clicked in response to them. Then search query text is matched to clicked web page text and “key” paths are identified. Jaccard sets similarity function is employed to compare web documents and later to cluster them.

1.3.4. URL Pattern Search Based Methods

All above proposed web page structural similarity measures are purely based on web page content analysis and comparison. However, most efficient and scalable web page clustering techniques depend on meta-data about pages, such as those based on web page URL analysis and pattern search (Blanco *et al.* 2011; Hernández *et al.* 2012). Our own proposed method belongs to this category. However, instead of comparing URLs we compare XPaths address of inbound-link in originating web document.

A good example from the latter category is the work done by Blanco et al. (Blanco *et al.* 2011). They show that, using only the URLs of web pages and simple content features, it is possible to cluster web pages effectively and efficiently. The proposed technique looks at URLs holistically and tries to detect repeated patterns in a set of URLs. This approach lets avoid pairwise comparison of URLs. Furthermore, to boost the clustering efficiency content features of web documents are incorporated. To be more specific, proposed method tries to identify template-related text occurrences in web pages, e.g. “Address:” and “Opening hours:” and some terms that are related to the data itself. Each term is saved with its XPath location in the HTML tree. This part of the method is indeed similar to “key” paths technique proposed by Chakrabarti et al. (Chakrabarti, Mehta 2010).

1.4. Web Data Extraction Systems in Enterprise Environment

The aim of this section is to review commercially available state-or-the-art web data extraction systems in context of online business intelligence. First of all, to remove any ambiguity from the term *online business intelligence* and to be more specific, we list concrete business scenarios and applications where online business intelligence is or can be practically employed. Then we review

commercially available state-of-the-art web data extraction tools that can be used for online business intelligence.

The main objective of online business intelligence is to collect valuable information from many web sources to support decision making and thus gain competitive advantage. However, the online business intelligence presents non-trivial challenges to web data extraction systems that must deal with technologically sophisticated modern web pages where traditional manual programming approaches often fail.

The web is full of data. Never before have we witnessed such constantly increasing and at first glance easily accessible repository of data about everything. The size of the indexable web, i.e. the web sites which are considered for indexing by the major search engines, is thought to be at least 11.5 billion pages as of the end of January 2005 (Pisa *et al.* 2005). Even more, 400 to 550 times larger amount of information resides in *Deep web* (Bergman 2001). The term *Deep web* refers to content hidden behind web forms. In order to retrieve such content a user has to interact with forms and perform a meaningful submission. The prime examples of Deep web are car advertising listings, real estate listings, statistical department databases and etc. Accessing information published on web sites (or Deep web) has been a long standing challenge in the data extraction community (Bohannon *et al.* 2012; Cafarella *et al.* 2011; Elmeleegy *et al.* 2011; Furche *et al.* 2012b; Madhavan, Halevy 2009).

Today's competitiveness dictates the need for business organizations to constantly seek timely and accurate information to support decision making and action. Such information seeking process is usually defined as *business intelligence* (Fleisher, Bensoussan 2003) and web is almost a perfect source for it. More specifically, the term *business intelligence* can be referred to as (Lönnqvist, Pirttimäki 2006):

1) Relevant information and knowledge describing the business environment, the organization itself, and its situation to its markets, customers, competitors, and economic issues
2) An organized and systematic process by which organizations acquire, analyse, and disseminate information from both internal and external information sources significant for their business activities and for decision making

Some other related terms include competitive intelligence, market intelligence, customer intelligence, competitor intelligence, strategic intelligence, and technical intelligence (Lönnqvist, Pirttimäki 2006). Business intelligence provides valuable information to companies in a timely and easily consumed way and enhances the ability to reason and understand the meaning of it through, for example, discovery, analysis, and *ad hoc* querying (Lönnqvist, Pirttimäki 2006).

When dealing with data coming from online sources, i.e. the Web, the term *online business intelligence* is used.

Even though there is enormous amount of data on the Web, researchers are studying the data extraction field for decades, businesses are striving for valuable data and are more than ready to pay for it, there is still no universal and easily deployable solution to fully leverage the data on the Web. Business intelligence systems seeking to collect valuable data from the Web must overcome many great challenges posed by the Web itself, such as finding appropriate data sources, determining their credibility, navigating technologically sophisticated web sites, submitting meaningful web form queries to retrieve Deep web content, extracting, cleansing, understanding and integrating data, and constantly dealing with heterogeneity in every step (Baumgartner *et al.* 2009b; Cafarella *et al.* 2011, 2008; Madhavan *et al.* 2007).

1.4.1. Applications for Web Data Extraction Systems

Given that most information on pricing, product availability, store locations, and so on, is available on the Web, *online market intelligence* is becoming the most important form of business intelligence (Baumgartner *et al.* 2009b). Market intelligence is the ability to understand, analyze and assess the environment of a firm with customers, competitors and markets, and industries, that conduces strategic planning and help decision making (Juntarung, Ussahawanitchakit 2008). Currently, almost every large retail company has online market intelligence needs for marketing and pricing (Baumgartner *et al.* 2009b). Here we list typical real world examples when businesses employ online business intelligence to monitor their market environment.

Background Check

Background screening is a profession that absolutely demands precision and timeliness. Erroneous background checks could result in stiff penalties as well as loss of business (Connotate 2012). By background checking companies are verifying the background of a customer or business partner. For example, background checking process may involve accessing courts' web sites in hundreds of jurisdictions: an automated data extracting program must query each Deep web court database to check if particular person or a company has any ongoing legal battles, or if it does have any active legal restrictions and etc. In addition to online court records, many professional associations post information on the Web. Online business intelligence can allow companies to aggregate the data from multiple sources to perform comparisons and verify the validity of credentials (Connotate 2012), such as certificates, honours and etc.

Competitive & Pricing Intelligence

A company's success hinges on identifying and responding to today's hyper competitive environment, especially in online settings. Company's challenge is to gather accurate competitive intelligence, analyse it and act as quickly as possible (Connotate 2012). Consider for examples the three scenarios presented by (Baumgartner *et al.* 2009b):

"An electronics retailer would like to get a comprehensive overview of the market in the form of a dashboard displaying daily information on price developments including shipping costs, pricing trends, and product mix changes by segment, product, geographical region, or competitor."

"A supermarket chain wishes to be continually informed about their competitors' product prices. Moreover, they want to be immediately informed in case a competing supermarket chain issues a special offer or promotion. They need to react very quickly to price changes or new special discounts in order to maintain their competitive position. They also want to be informed as soon as new products show up on the market."

"An online travel agency offering a best price guarantee needs to know at which prices the packages they offer are sold over the Web by competing travel agencies. Moreover, they wish to be informed about the average market price of each travel product they feature."

Compliance & Risk Management

Companies may want to be informed of updates on sanctions lists and regulations at the international, federal and state levels. Automated online business intelligence makes it easier to ensure compliance with laws regulating rogue nations or organizations financing, ascertain the legal integrity of potential business partner, and reduce exposure to financial fraud and identity theft (Connotate 2012). For example, a company working in car or car-parts trading business may want to check automatically if a particular vehicle or its part is not included in stolen property registers around the globe. Cost-effective management of compliance and risk is a complicated challenge, because precise source monitoring and Deep Web querying must be executed on-the-fly: all the automatically accessed data should be immediately extracted, cleaned, integrated and presented.

Customer Sentiment Analysis

Today many customers are buying online and publicly sharing their user experience, opinions and buying preferences. In most cases users express their opinion as comments, forum or social media post, tweets and etc. Analysing

customer sentiment is fundamental to maintaining a competitive edge in the delivery of goods and services (Connotate 2012). Online business intelligence solutions should be able to access Facebook, Twitter, or any other social media web site, automatically identify posts about particular product, extract text, execute natural language processing and understand the sentiment. The same data extraction process can also be applied to hundreds of other sources, such as blogs, online forums, product review sites, YouTube and etc.

News & Content Aggregation

Media monitoring companies aggregate news articles and comments from many online web sites. It is not a trivial problem to monitor hundreds of online sources, which are usually heterogeneous in style, news format, and navigation. Furthermore, industry leading media monitoring companies should be able to automatically classify collected news articles by their topic and group articles describing the same event. Brand name mentioning monitoring is also an important task, which should be executed on thousands of news articles. A proper online business intelligence solution should offer scalable, automated technology to crawl and extract data from hundreds of thousands of news sites, archives and corporate web sites.

Financial Data Aggregation

According to (Connotate 2012) “every moment of every day, political events, financial filings, corporate actions and many other market-moving events are posted on the Web. Detecting and communicating these events to the financial community in near real-time is essential to building and maintaining market share in the world of financial data. Speed and accuracy are paramount”. New or updated financial data appears every day in a big variety of online sources, such as government data portals, news articles, companies’ news feeds, even on social media. For example, hedge funds involved in algorithmic trading monitor thousands of sources in real-time to immediately detect breaking news and swiftly sell or buy particular stocks. Even such trivial data as weather temperature is monitored across the globe and in event of unexpected drop in temperature the stocks of oil, gasoline, electricity or heaters manufacturing companies can be bought in a matter of seconds. With the increasing wealth of information and content available on the Web, the opportunity to use it for timely notification, analysis and enhanced decision making is unprecedented (Connotate 2012).

1.4.2. A Typical Commercially Available Web Data Extraction System

(Baumgartner *et al.* 2009a) define web data extraction system as "a software system that automatically and repeatedly extracts data from web pages with changing content and delivers the extracted data to a database or some other application". They further divide web data extraction tasks to five functions (Baumgartner *et al.* 2009a):

1. Web site interaction, which includes mainly the navigation to usually pre-determined target web pages containing the desired data.
2. Support for wrapper generation and execution, where a wrapper is a program that identifies the desired data on target pages, extracts the data and transforms it into a structured format.
3. Scheduling, which allows repeating data extracting tasks by constantly revisiting target web pages.
4. Data transformation, which includes filtering, transforming, refining, and integrating data extracted from one or more sources and structuring the result according to a desired output format (usually XML or relational database tables).
5. Data provision, which is delivering the extracted structured data to external applications such as databases, data warehouses, business intelligence systems, decision support systems and etc.

See Figure 1.7 where the architecture of a typical state-of-the-art web data extraction system is presented.

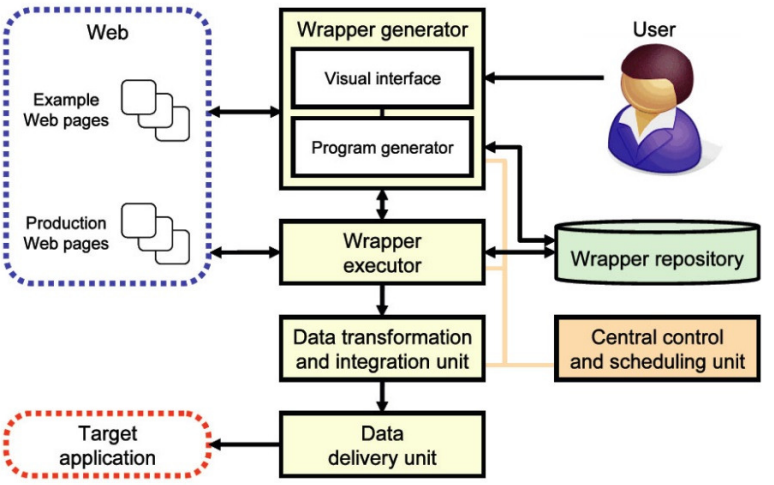


Fig. 1.7. The architecture of a typical state-of-the-art commercial web data extraction system (Baumgartner *et al.* 2009a)

In the system the wrapper generator helps the user to build data extraction rules. It usually has a visual interface displaying rendered target web pages and the user is asked to visually mark which data in a web page should be extracted. The subunit that automatically generates the wrapper (data extraction rules) referred to as the program generator. This module interprets the user actions on the example web pages and successively generates the wrapper. The navigation or a Deep web form submission in a target web site can be recorded and later automatically reproduced. The wrapper runs previously generated wrappers, which are stored in wrapper repository. The data transformation and integration unit cleans, combines, transforms and integrates extracted data. The data delivery unit delivers data via appropriate channels such as FTP, HTTP, E-mail and etc.

Although scientific literature is full of different approaches and proposed systems (Crescenzi 2001; Zhai, Liu 2006; Zhao *et al.* 2005) (see also a survey by (Ferrara *et al.* 2012)) to extract data from web pages only a few of those techniques are built into commercially available products. One of the most prominent examples of such systems coming from an academic research field is Lixto (Baumgartner *et al.* 2009b). With *the Lixto Visual Developer* software, wrappers are created in an entirely visual and interactive fashion. See Figure 1.8 where a screenshot of *Lixto Visual Developer* user interface is presented. In the middle there is a fully functional Mozilla web browser with loaded target web page. On the left navigation steps are recorded. These steps may include submitting a form, clicking on menu items, following page navigation and etc. In the bottom there are options to configure extraction rules

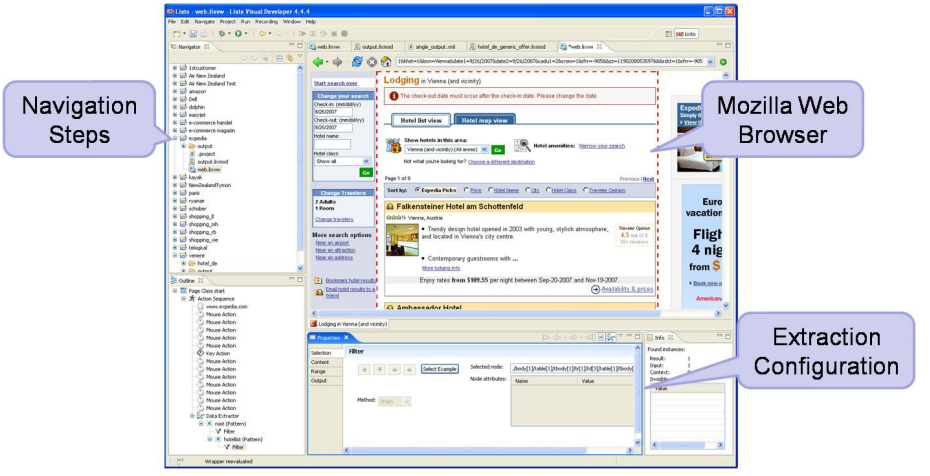


Fig. 1.8. The visual interface of the commercially available Lixto web data extraction system (Kannan 2010)

1.5. Research on Data Extraction in Lithuania

Data extraction in general and other closely related research fields, such as information extraction from free text, knowledge extraction, and etc., has received attention from Lithuanian scientists and business entities.

For instance, (Krilavičius *et al.* 2012) proposed a method to automatically monitor and classify Lithuanian news media articles by identifying and extracting main topics and facts, such as locations, names, citations and etc. They demonstrated that a combination of information retrieval and natural language processing tools with appropriate changes can be successfully applied to Lithuanian media texts. A similar technique with an additional machine learning module has been also applied to create a system that extracts information from the texts of police event summaries (Kaušas *et al.* 2010) .

(Normantas, Vasilecas 2012) proposed a method for extracting business rules from existing enterprise software systems. Their approach facilitates software comprehension by enabling traceability of implementation of business rules and business scenarios in the software system. The same authors have also reviewed (Normantas, Vasilecas 2013) many methods for business knowledge extraction from existing software systems.

The problem of extracting information from business systems and processes is also addressed by (Skersys *et al.* 2013) where they propose an approach for extracting business vocabularies from business process models. Similar problem is studied by (Paradauskas, Laurikaitis 2006) were they analysed the process of enterprise knowledge extraction from relational database and source code of legacy information systems.

(Laukaitis, Vasilecas 2008) presented a system that leverages extracted information from an online linguistic resource (*Wordnet*[10]) to induce syntactic and semantic transformation rules. These rules are then used to improve machine learning based language translation.

(Damaševičius 2009) researched a way to automatically construct ontology. They proposed a method to leverage extracted information from web search results for learning and generating domain concept hierarchies. Their method is based on generating derivative features from web search data and applying the machine learning techniques.

Daudaravičius participated in a common work with (Henriksson *et al.* 2006) and demonstrated how synonyms of medical terms can be extracted automatically from a large corpus of clinical text using distributional semantics.

[10] http://wordnet.princeton.edu/

Aside from purely academic works, a private Lithuanian company called TokenMill[11] is applying information extraction, natural language processing, text analytics and machine learning techniques in order to solve real life big data and text analysis problems.

1.6. Conclusions of Chapter 1 and Formulating Tasks for the Dissertation

1. The literature review revealed that a typical contemporary web site stores data in an underlying relational database. Upon a web page request, the data is retrieved from the database and embedded into the requested page using some fixed template. Such web pages are known as template-generated, and the embedded data is called structured data. Data extraction may be viewed simply as a reverse engineering task. That is, given the HTML mark-up encoded data (i.e., web pages), the extraction process should identify the original data model and extract the data from the HTML source code.
2. Visual and structural regularity is found among web pages coming from same web site where pages are template-generated. This regularity is very important feature for structured data extraction systems, since it provides clues how the original data from a database is encoded into HTML code. Thus the key tasks for structured data extraction from template-generated web pages become:
    a) Detecting visual and structural regularity within a web page;
    b) Identifying embedded structured data;
    c) Constructing data extracting wrapper;
    d) Reusing the same wrapper on structurally similar web pages.
3. Extracting structured data from many different web sources is typically a process requiring human effort that brings considerable costs to the organizations. It is also assumed that it is infeasible that any manual human work requiring structured data extracting approach could be successfully applied to extract data at Web-scale. So, much of current research is directed into developing fully-automatic structured web data extraction techniques that could be applicable to extract data at Web-scale.

Taking into consideration what is said, the following issues should be pursued:

[11] http://www.tokenmill.lt/

1. A proposal is needed for a new automatic structured web data extraction method that would be capable of extracting data from visually and structurally heterogeneous and technologically sophisticated modern template-generated web pages.
2. The said method should exploit both visual and structural regularities of template-generated web pages and be aimed to extract data from list pages, i.e. those data rich web pages where more than one embedded data record is present.
3. The said method should generate a wrapper that could be later reused to extract data from same template pages.
4. A proposal is also needed for an efficient method to cluster structurally similar template-generated web pages.
5. Publicly available benchmark datasets should be identified to test the said methods. In case public dataset are not available, an appropriate datasets should be created.
6. The said methods should be tested and compared to other state-of-the-art techniques.
7. The said two methods should have a potential to be integrated into Web-scale structured data extraction systems.

2

A Method for Extracting Structured Data from Template-Generated Web Pages

In this Chapter we introduce a novel method for extracting structured data records from template-generated web pages. The proposed method is called ClustVX (derived from *Clustering Visually similar XPaths*). The method is based on clustering visually similar web page elements. It first renders given web page in a contemporary web browser, then clusters visually and structurally similar repeating web page elements to identify the underlying pattern of embedded structured data records.

The results presented in this Chapter are published in (Grigalis, Čenys 2014a), (Grigalis 2013), (Grigalis *et al.* 2012b).

2.1. Data Extraction from Web Pages

The presence of vast structurally and visually heterogeneous web sources pose key challenge to web search today (Madhavan *et al.* 2007). Web pages with structured data are easily understandable by humans, but automatically extracting

the same data by computers at Web-scale is a very difficult task (Baumgartner, Flesca 2001; Cafarella, Halevy 2009; Cai *et al.* 2003).

Many solutions are proposed to extract structured web data. They typically search for repeating patterns in a web page by calculating the similarity between HTML tag tree nodes (Álvarez *et al.* 2008; Jindal, Bing 2010; Kayed, Chang 2010; Su *et al.* 2011; Zhai 2005). However, such methods do not show consistent results on different benchmark datasets and are prone to errors when dealing with contemporary WEB 2.0 pages. That happens, because modern web browsers have very high tolerance for an invalid HTML code and thus a lot of web pages do not obey the W3C HTML specifications. Incorrect HTML code leads to error in constructing HTML tree, which, in turn, hinders structured data extraction process.

Moreover, HTML tree was initially introduced for more readable HTML presentation in the browser rather than description of the semantic data structures in the web pages (Cai *et al.* 2003). Widespread invalid HTML code especially hinder web data extraction from sophisticated WEB 2.0 pages, where HTML tree is often dynamically modified by various JavaScript codes, new data is added by asynchronous requests to web server and elements are positioned with Cascading Style Sheets (CSS). The underlying structure of most modern web pages is complicated more than ever before and has very weak ties to their visually rendered versions displayed on modern web browsers (Liu *et al.* 2010). It is becoming very difficult to successfully extract structured data from web pages by just analysing raw HTML code which is retrieved directly from a web server. Although there are already some works (Liu *et al.* 2010; Nie, Wen 2007; Zhai, Liu 2006; Zhao *et al.* 2005) that utilize the visual features of web pages to extract data, but these methods often lack proper experimental evaluation on publicly available benchmark datasets or have some limitations which we address in details in related work Section.

In this Chapter we introduce a conceptually different approach, called ClustVX (Clustering Visually similar XPaths). ClustVX is based on two fundamental observations.

First, vast amount of information on the Web is presented using fixed templates and filled with data retrieved from underlying databases (Cafarella *et al.* 2011). For example, Figure 2.1(a) shows three data records with structured data describing three digital cameras in an online store. The three data records are listed according to some unknown to us visual style and the data comes from an underlying database. All three data records are structurally similar and are placed in one region of a web page (*data region*) (Zhai, Liu 2006). Since all these data records are placed in one place, each of them has almost the same XPath, i.e. the tag path from the root node in HTML tree to the particular web page element.

Second, web page designers optimize visual presentation of structured data records for the human reader. So although the templates and underlying data differ from site to site, humans understand them easily by analysing repeating visual patterns on a given web page (Miao *et al.* 2009). The data that has the same semantic meaning is often visualized using the same style (Liu *et al.* 2010). Therefore humans, viewing a web page, are able to comprehend its unique structure quickly and effortlessly and distinguish each data record and its unique attributes, such as photos, titles, prices and etc. For example, in Figure 2.1(a) prices brown red and bold, title is green and bold, text "Online Price" is grey and in normal text.

a) an example of three digital cameras (data records) in a web page

Xstring:	htmlbodydivdivdivfonta-Verdana,brown-red;400		
$84.95	/html/body/div[3]	/div[1]	/div/font/a
$174.95	/html/body/div[3]	/div[2]	/div/font/a
$84.95	/html/body/div[3]	/div[3]	/div/font/a

b) a cluster with visually similar price elements

Image 1	Samsung ES80	$84.95	Online Price
Image 2	Fujifilm FinePix T300	$174.95	Online Price
Image 3	Vivitar ViviCam F529	$84.95	Online Price

c) desired extraction results

Fig. 2.1. An example of structured web data extraction with ClustVX method

ClustVX exploits both observations by representing each web page element with a combination of its XPath and rendered visual features such as font size,

font colour and etc. Getting visual features of web page elements is not a trivial task. It is not enough to simply extract *style* or *class* attributes from HTML code. The information describing the visual style of a web page element can be stored in many different locations, such as in Cascading Style Sheet (CSS) files, in attributes, in parent elements and etc. Furthermore visual appearance of an element can be modified on-the-fly by JavaScript code. Correct web page visual rendering can be only achieved by modern web browsers and their complex rendering engines. ClustVX method employs a modern Mozilla Firefox web browser to visually render web pages and to retrieve resulting rendered visual features of web page elements. For each visible web page element we encode XPath and visual data into a string called Xstring. Clustering Xstrings allows us to identify visually similar elements, which are located in the same region of a web page and in turn have same semantic meaning. See Figure 2.1(b) where price elements are clustered together according to their Xstring. Subsequent data extraction leads to a machine readable structured data Figure 2.1(c).

Structured data extraction process with ClustVX method is also based on two structural observations about data record representation in a web page. First, a group of data records are usually rendered in a contiguous region of a web page (Zhai, Liu 2006) and are visually similar. Second, a group of data records are formed by some child sub trees and at some level have same parent node (Zhai, Liu 2006). Thus, by calculating longest common prefix of XPaths from each cluster of visually similar web page elements, we can find the exact locations of data records groups (data regions) in a page. For a simple example, consider the Figure 2.1(b), where XPaths of clustered price elements are located. First, we find the longest common prefix *(/html/body/div[3]*) of these clustered XPaths. The prefix leads us to the particular region of a web page, where data records are located. Then, the longest common suffix *(/div/font/a*) is data items' path in the data record. The XPath substring between prefix and suffix (*/div[*]*) is used to segment data region into data records. All clusters that have the same longest common XPath prefix present one particular data region. If there are many data regions in one page, ClustVX locates them all.

2.2. The Proposed Method

In this Section we in detail explain the process of structured web data extraction with the proposed ClustVX method. We begin by reviewing structural and visual features of web pages that are used in structured data extraction process. The process itself consists of many separate steps. First of all HTML of web pages is pre-processed to enclose unenclosed text tokens and to embed visual information to each HTML element. Then each visually visible text element of a

web page is clustered to clusters according to its visual similarity and XPath strings. By manipulating XPaths in clusters ClustVX locates data regions, segments data records. A *visual weight* for each data region is then calculated to determine its importance in a page. And only then data records are extracted. The final steps of structured web data extraction process according to the proposed ClustVX method are wrapper induction and data items extraction.

2.2.1. The Architecture

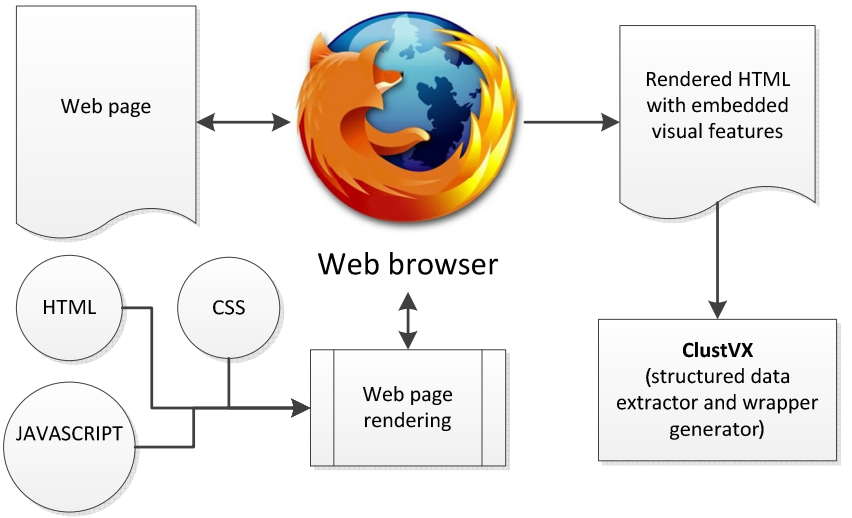


Fig. 2.2. The architecture of the proposed method implemented as a system

The general architecture of proposed ClustVX method is presented in Figure 2.2. In the context of a typical commercially available web data extraction system, as defined by (Baumgartner *et al.* 2009a) and seen in Figure 1.7, the prototype system should be generally considered a wrapper generator and a wrapper executor.

In the system, a Web browser is used to download HTML code from a web server and fully render web page in a browser window (in current implementation of ClustVX method we use Mozilla Firefox web browser). Web browser downloads all required additional data, such as cascading style sheets (CSS), JavaScript code and etc. In the browser all retrieved JavaScript code is executed, CSS is used to visually style web page elements. All this process visually renders web page and prepares it for browsing. After web page is rendered at browser level we artificially execute additional JavaScript code to retrieve visual features

and only then pass resulting HTML code to ClustVX method. Please see Figure 2.3 where detailed activity diagram is presented.

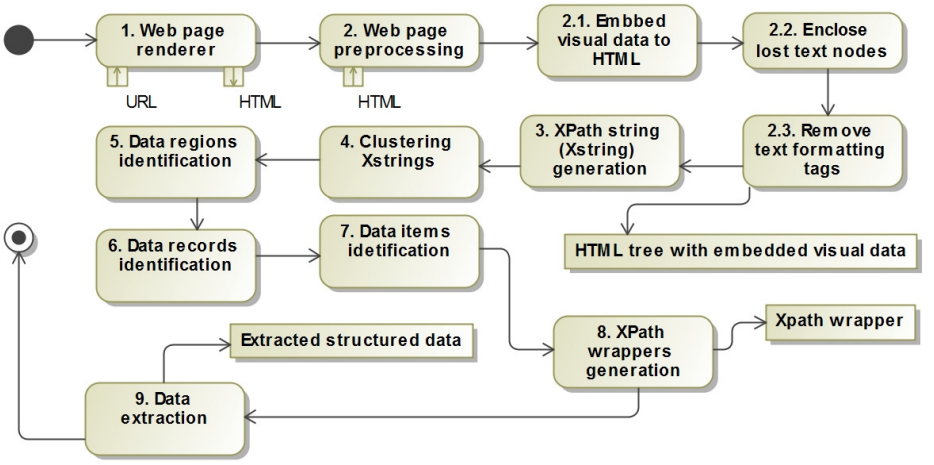


Fig. 2.3. Activity diagram of structured data extraction with ClustVX

As we further see from Figure 2.3, after rendering web page we proceed to web page pre-processing stage where a JavaScript code is injected into the web page to select rendered visual features, such as font size, font style, font colour and etc. The extracted visual features of each HTML element node are then embedded into the element node as attributes. Unenclosed text nodes are put under artificially created elements. In this stage we also remove all HTML text formatting elements, such as , <bold>, <italic> and etc. The HTML tree with embedded visual data, enclosed text tags and removed text formatting element nodes is passed to *Xstrings* generation stage. Xstrings are modified XPath location strings with added visual data.

Clustering stage clusters Xstrings into separate clusters. Data records extractor uses from clustering stage inherited information and identifies data regions available on the web page. Each data region gets rules (XPaths) to extract data records, which are inside data region. Then data items extractor extracts and aligns all data items within each data record.

The results of the data extraction process are visualized in a HTML table. During the data records extraction and data items alignment phases XPath wrapper is automatically induced. XPath wrapper can be later reused to directly extract structured data from the web page without a need to repeat all the stages. Detailed explanation of each extraction step is available in the following Sections of this Chapter.

2.2.2. Exploiting Structural and Visual Features of Web Pages

It is becoming very difficult to access and extract structured data from web 2.0 pages, where many parts of page content are generated dynamically with JavaScript code. This leads to a necessity to first of all render a web page in a contemporary web browser which executes all embedded JavaScript codes, applies cascading style sheets and etc. Only then we can leverage visual features of a web page, such as font colour and font size for structured web data extraction. In this Section the main visual and structural features of a web page are presented. We exploit these features to extract structured web data.

As we know, an HTML document can be represented as tree data structure. The modelled HTML tree is widely used in structured data extraction algorithms (Bohannon *et al.* 2012; Chang, Kuo 2004; Chang 2001; Dalvi, Bohannon 2009; Myllymaki, Jackson 2002), where usually dynamic programming is employed to find similar branches of the tree. Typically, the similarity between two web pages is determined using variations of tree or string edit distance algorithms. Contrary to these approaches based on comparison, ClustVX employs clustering of location paths (XPaths) which in are also enhanced with visual features of web page elements.

There are two kinds of XPath: relative location XPath and absolute location XPath. A relative XPath consists of a sequence of one or more location steps separated by /. The steps in a relative location path are composed together from left to right. Each step in turn selects a set of nodes relative to a context node. Relative XPath can improve the robustness of extraction wrapper (Dalvi, Bohannon 2009). However, relative XPath is not used in ClustVX algorithm. Instead we exploit the absolute XPath, which lets us better to compare HTML elements and find similar ones. An absolute XPath consists of / optionally followed by a relative location path. A / by itself selects the root node of the document containing the context node. If it is followed by a relative location path, then the location path selects the set of nodes that would be selected by the relative location path relative to the root node of the document containing the context node (Clark *et al.* 1999).

XML documents operated on by XPath conform to the XML Namespaces Recommendation[12]. The document tree contains nodes. There are 7 types of nodes, according to the Recommendation:

1. Root nodes.
2. Element nodes.
3. Text nodes.
4. Attribute nodes.

[12] http://www.w3.org/TR/REC-xml-names/

5. Namespace nodes.
6. Processing instruction nodes.
7. Comment nodes.

In data extraction process ClustVX uses four kinds of nodes: root nodes, element nodes, text nodes and attribute nodes. Root node is regarded as the topmost location of the HTML tree and is accessed every time we start evaluating and XPath expression. Element nodes are the main nodes, which contains data items. There is an element node for every element in the document. The task of structured data extraction is to find data regions containing similar sets of data items (element nodes) group them into the sets (data records) and extract. Each element node has an associated set of attribute nodes, where the element is the parent of each attribute node. However, an attribute node is not a child of its parent element (Clark *et al.* 1999). In the web page pre-processing stage we gather many visual clues about each element in the page and embed that information to the attribute nodes of each particular element.

HTML code together with images and visual style information is rendered in a web browser. The whole rendering process consists of computation of style data, frames construction, constant reflowing to represent changes or adding newly downloaded information. The result is a web page as we see it in a web browser window. It has been demonstrated that computed visual information can improve data extraction process (Liu *et al.* 2010). In this Section we describe the two main visual features of a fully rendered web page, which helps ClustVX to extract data: web page layout and web page element font features.

Each web page element, such as let it be an image, a text string, or a white space, after web page rendering is placed to some location. The whole web page can be seen as a coordinate system where x axis describes horizontal position and y axis describes vertical position. The most left top corner of a web page is a starting point, where x and y are equal to zero. The more an element is down in the page the bigger y value it has. And similarly, the more an element is to the right in the page the bigger x value it has. According to W3 specification (Kesteren 2011), each web page element is bounded by a rectangular box which is called bounding rectangle. The left top corner of a bounding rectangle is positioned at an exact coordinate (x, y) in a web page. So the left, top, right and bottom properties are describing the bounding rectangle, in pixels, with the top-left relative to the top-left of the page view. Combined view of all rectangles makes a web page view. The ClustVX method exploits these visual attributes to determine visual position and size of each element.

Each rectangle box, according to W3 specifications (Kesteren 2011), has the following attributes:

1. Float top.
2. Float right.

3. Float bottom.
4. Float left.
5. Float width.
6. Float height.

So the left, top, right and bottom properties are describing the bounding rectangle, in pixels, with the top-left relative to the top-left of the page view. Combined view of all rectangles makes a web page view. The ClustVX method makes use of these visual attributes to determine visual positions and sizes of web page elements.

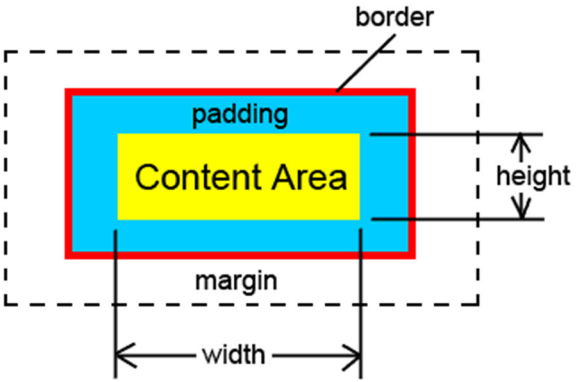


Fig. 2.4. Visual box model (Beach 2013)

In a rendered web page each visible HTML element has a rectangular bounding box (see Figure 2.4). This box determines the spatial relations among web page elements. In clustering process of ClustVX method we exploit the attributes of rectangular bounding box to determine the visibility of web page elements. This is important, because we are only interested in visible web page elements.

A text element in a web page can be styled in many different ways (Table 2.1). For example, it is possible to set individual size, colour, weight, strikethrough for each text string. The change of any of the font features, such as *size, face, colour, strikethrough, weight, italic, underline,* modifies the visual presentation style of the text element. Visual font features are often used to differentiate one semantic type of text from another. These features help ClustVX method to efficiently group semantically similar text elements to same clusters.

Table 2.1. An example of visual font features

Attribute name	Attribute example	Rendered Text example
size	15pt	text
face	Courier New	text
color	red	text
strikethrough	boolean: true	~~text~~
weight	strong	**text**
italic	boolean: true	*text*
underline	boolean: true	text

2.2.3. Web Page Pre-processing

After a web page has been rendered in a Mozilla Firefox browser, but just before the extraction process is started, the web page is modified to enhance web data extraction and to speed up whole process. The modification includes: a) embedding visually significant data to the attributes of each web page element; b) enhancing HTML tree structure by enclosing hanging text nodes within new parent node; c) removing any text formatting. Those steps are described in the following Sections.

```
<span
left="631.5"
top="1477.5"
width="195"
height="17"
fontdata="Arial,Verdana,Helvetica,sans-serif;rgb(0, 0,
0);10px;normal;400;none" class="artbox_1col_finance">
```

Fig. 2.5. An example of visual data embedded into elements' attributes

The technique embedding visual data into HTML code is employed to enable processing web page source code without constant API calls to a browser and, at the same time, retaining accessibility to all visually important information. It is done in this way: while a web page in rendered in a browser window all important visual information is extracted using browsers API calls and JavaScript code. The extracted information is then embedded into each corresponding HTML element as an additional attribute. These attributes has no effects on visual appearance of a web page. Figure 2.5 shows source code of HTML element with embedded visual information to "*left*", "*top*", "*width*", "*height*", and "*fontdata*" tags. These

visual features are later used in structured data extraction stage of ClustVX method. Visual appearance features, such as *font type, font size, font colour* and visual position features of rectangle boxes, such as *left, top, width, height*, are embedded into HTML code.

Some web pages have text strings (text nodes), which has no individual HTML element as a parent. Those nodes are often put under the same parent together with other HTML elements and other text strings. Because ClustVX method identifies the exact position of each HTML element by using XPath, each text string must have its own HTML element and no other sibling under the same XPath should exist. So in web page pre-processing stage we search for each unenclosed (*hanging*) text strings in HTML tree and enclose them with tag. Figure 2.6 demonstrates an example of text strings in HTML tree, which do not have individual enclosing tag and will be put under the element. The unenclosed text nodes are showed in bold on left. On right side of Figure 2.6 we see the resulting HTML source code after the enclosure process. The process added additional enclosing tags shown in red colour.

```
<p>
  <b>
    Some unenclosed text
    <br>
    E-Mail Address:
    <a
href="mailto:K6@hm.com">
    <i> K6@hm.com </i>
    </a>
    <br>
    Contact By: E-Mail
    <br>
    Ad Number: 101205
  </b>
</p>
```

```
<p>
  <b>
    <span>Some unenclosed text</span>
    <br>
    <span>E-Mail Address:</span>
    <a href="mailto: K6@hm.com ">
    <i> K6@hm.com </i>
    </a>
    <br>
    <span>Contact By: E-Mail</span>
    <br>
    <span>Ad Number: 101205</span>
  </b>
</p>
```

Fig. 2.6. An example of enclosed hanging text nodes

HTML elements that are used to visually style text, such as <*em*> to emphasize, <*u*> to underline and etc. often do not help to identify structural nature of encoded text data. Contrary, we have observed that sometimes formatted text nodes can impede data extraction process. Often text formatting nodes divide semantically contiguous text into two or more text nodes in HTML tree. As seen in Figure 2.7, formatting element node (<*b*> - bold) divides a description of a book into three text nodes. It is semantically contiguous text and should not be divided. This kind of division impedes the data extraction process, so we simple remove those formatting element nodes and form a new contiguous text node.

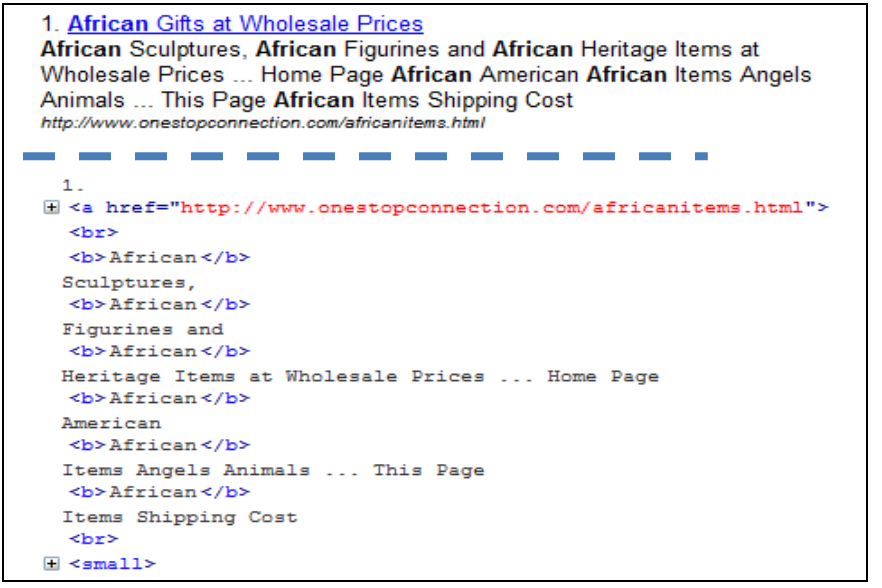


Fig. 2.7. An example of semantically contiguous text divided by text formatting tags

2.2.4. Generating XPath Strings With Visual Data (Xstrings)

In this stage ClustVX algorithm iterates each HTML element. For each visible web page element we generate an absolute location path (XPath). Furthermore, we extract all embedded visual data, which is stored in the attributes of that element: font size, font style, font colour, font weight, font strikethrough. For example, let's say we have obtained font data and XPath of the element. The next step is to remove from XPath the indexes, separation slashes and leave only tag names. Then we join tag names into a string together with font data. We call resulting string an *Xstring*. We will use the Xstrings in the next clustering stage of the ClustVX algorithm. See Table 2.2 for an example with Xstring formation.

2.2.5. Clustering Visually Similar Web Page Elements

To ease the understanding of structured data presented in a web page, web page template designers often arrange the data records and the data items with visual regularity to meet the reading habits of human beings. Data items of the same semantic in different data records are similar on layout and font (Liu *et al.* 2010). By exploiting this observation the clustering stage of ClustVX web data extraction method puts visually similar web page elements into the same clusters.

The similarity of elements is computed by comparing tag paths and font data, i.e. the Xstrings. To cluster according to similarly, as in VIDE (Liu *et al.* 2010), a single one pass clustering algorithm is employed: we take any first HTML element from a HTML tree and look at its visual appearance data (Xstring). If there is no cluster representing such Xstring we simple create a new cluster and put the element there. If element has the same visual properties as elements from other cluster, we put that element there. In other words, if two elements have same Xstring string, they are put into the single cluster. The same thing is done with all remaining elements from HTML tree.

Table 2.2. An example of Xstring formation

XPath:	/html[1]/body[1]/div[2]/div[1]/div[1]/div [5]/p[2]/a[1]
Font Data:	Arial,Verdana,Helvetica,sans-serif; rgb(0, 0, 0); 10px; normal; 400; none
XSTRING as a result:	htmlbodydivdivdivdivpa-Arial,Verdana,Helvetica,sans-serif;rgb(0, 0, 0);10px;normal;400;none

2.2.6. Data Records Identification and Extraction

The objective of structured web data extraction is to extract all data records from a given web page. We assume that the web page is dynamically generated and hence an underlying template exists. Data items are usually located in data records, which all together make a data region. According to (Liu *et al.* 2010), an ideal structured data extractor should achieve the following: 1) all data records in the data region are extracted and 2) for each extracted data record, no data item is missed and no incorrect data item is included.

Data record extraction with ClustVX method is based on two observations about data records presentation in web pages: 1) A group of data records are usually rendered in a contiguous region of a web page (Zhai, Liu 2006) and are visually similar; 2) A group of data records are formed by some child sub trees and at some level have same parent node (Zhai, Liu 2006).

There exist two ways of embedding data records into HTML tree. For example consider that a single data record in a web page consists of three node tags *<A>, <DIV>* and *<SPAN>*. *<A>* is the first node in a data record and <SPAN> is a last. Figure 2.8 demonstrates two ways, how a list of mentioned data records can be presented in a web page:

1. Each data record belongs to only one parent node *(<DIV>$_1$, <DIV>$_2$,* and *<DIV>$_3$*).

2. A set of nodes forms a data record and they all are placed under the same parent. Here only one <*BR*> tag node separates the three data records.

While these two types of data records visually may be rendered the same, but as we see in Figure 2.8, the parent and children relationships in the HTML tree are totally different.

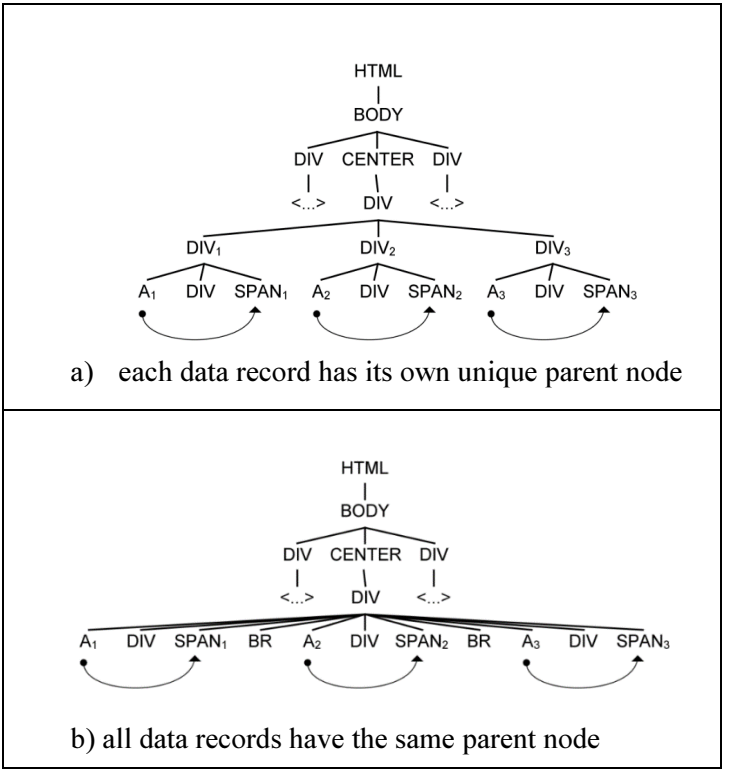


Fig. 2.8. Two types of data records structural representation in HTML tree

This difference has an profound implication for data records identification and separation: a) if there exist one parent for each data record, then we identify the boundary of data record just by looking at the boundary of the parent; b) if data record consists of a set of nodes, and all nodes of all data records in particular data region are under the same parent node, we first need to find the boundary of each data record and combine all nodes which belong to that data record into the same group. The detailed explanation of the process is described in following Section.

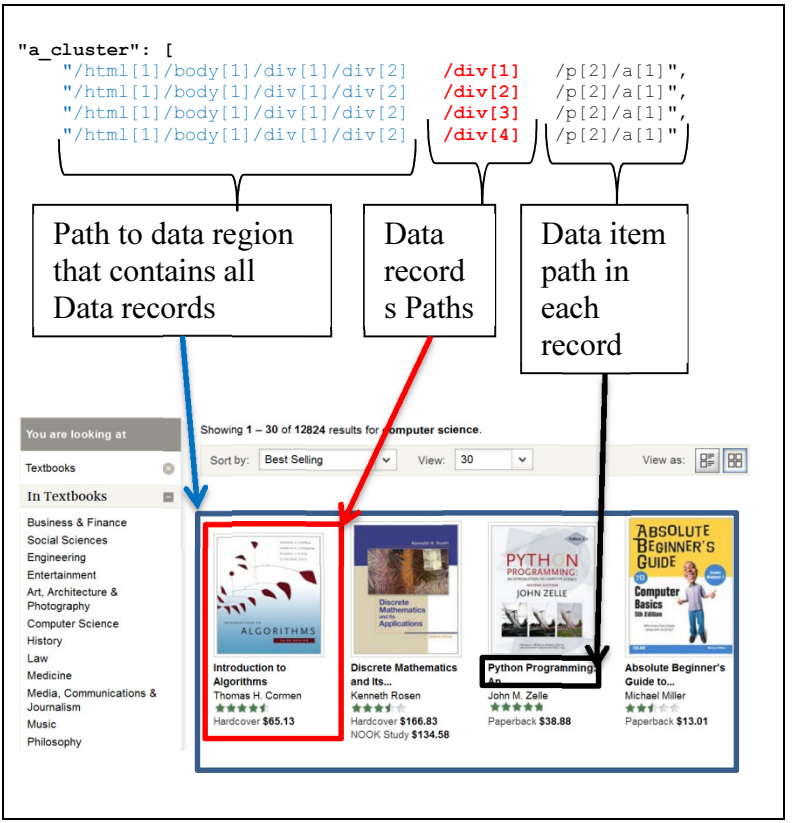


Fig. 2.9. Using cluster to find data region, data records and data items

2.2.7. Finding Data Regions, Data Records and Data Items

Each clustered HTML element has a unique XPath string, which points to the exact location of the element in the HTML tree. Using this information it is very easy to find data region, data records and data items in a web page. Just by calculating longest common tag path prefix of all elements in particular cluster, we find the tag path of a data region.

Depending on the type of data record, the two heuristics (for a visualization see Figure 2.10) are used to segment data records: 1) If data records are of a type A as seen in Figure 2.8(a), that is – each data record in a data region is under its own parent, then data records tag path will be the first tag path whose number differs in each clustered data item. Sometimes data record tag path can consist of two or more tag names. 2) If data records are of type B as seen in Figure 2.8(b), that is – each data record consists of a set of nodes and all data records and their

sets of nodes are under the same parent, then a different approach is used, which we call HTML tree hopping technique.

The tree hopping technique works in this way: ClustVX identifies the first (visually) data item in the first record of data region. In Figure 2.11(a) the first data item is "1: Amarillo Globe-News: Headlines". Then all other data items, which are visually located between the identified visually first one and the second, forms a data record. In Figure 2.11(a) the start of a second data record is "2: On The Edge Of Common Sense". So these three data items "1: Amarillo Globe-News: Headlines", "http://www.amarillonet.com/1/index.html", and "(Score 84, Size 18K, Last modified Nov-07-03)" belong to the data record 1. As we see, in the corresponding HTML source code in Figure 2.11(b), each data record is separated by "*

*" separator and all data records are under the same parent node.

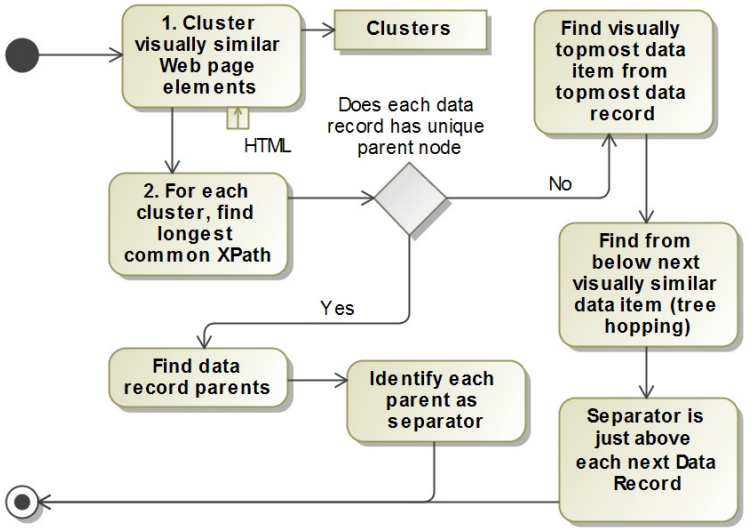


Fig. 2.10. The diagram depicting data records segmentation activity

Thus by manipulating and combining tag paths of the HTML elements in clusters we find tag paths of all data regions, data records and data items. See Figure 2.9 for a simple example.

2.2.8. Finding Visual Weights of Data Regions

Since there can be many data regions in a web page, it is handy to somehow rank these regions according to their importance. One straightforward way to do it is to calculate the visually occupying place of each data region (*visual weight*) in a rendered web page. We presume, that the more visual place the data region occupies in a rendered web page, the more important it is. Furthermore, the amount of data records and data items in that region also contributes to its visual weight.

During the visual weight calculation we square the number of data items to significantly improve their importance. Sometimes data regions contain only a few data records which are spread very widely on a page. Such data regions contain small amount of data and typically can be ignored. An example of widely spread data records is pagination numbers which are located above and below paginated data. In that case, even though the total occupying visual area is significantly big, but it is very widely spread and contains a small amount of textual data.

To decrease the visual weight of widely spread data regions and to increase the visual weight of regular data records with many data items we square the latter number. So with ClustVX method we use the following formula to calculate visual weight of particular data region, where VW is visual weight, DR_a is the area in pixels of data region, $|DataItems|$ is number of data items per data record:

$$VW = DR_a * |DataItems|^2 . \quad (2.1)$$

2.2.9. Extracting Data Records

Data record extraction process is very straightforward: since we already know the tag paths of each data region in a web page and the tag paths of data records, the only thing we need to do is to traverse the HTML tree and collect relevant information.

As described in previous Section on this Chapter, by analysing tag paths in every cluster of visually similar web page elements we derive tag paths of data regions, data records and data items.

Search found 25 documents that match your criteria:

1: Amarillo Globe-News: Headlines
http://www.amarillonet.com/1/index.html
(Score 84, Size 18K, Last modified Nov-07-03)

2: On The Edge Of Common Sense
http://www.amarillonet.com/2.index.html
(Score 82, Size 31K, Last modified Nov-07-03)

(a) rendered view in a browser

Data record 1	`<font size="2" face="arial,helvetica,sans-serif">` `<b>1: Amarillo Globe-News: Headlines</b>` `</font>` `    ` `<font size="1" face="arial,helvetica,sans-serif">` `<a href="http://www.amarillonet.com/1/index.html">` `http://www.amarillonet.com/1/index.html` `</a>` `</font>` ` <i>(Score 84, Size 18K, Last modified Nov-07-03)</i>`
Separator	**`  `**
Data record 2	`<font size="2" face="arial,helvetica,sans-serif">` `<b>2: On The Edge Of Common Sense</b>` `</font>` `    ` `<font size="1" face="arial,helvetica,sans-serif">` `<a href="http://www.amarillonet.com/2/index">` `http://www.amarillonet.com/2.index.html` `</a>` `</font>` ` <i>(Score 82, Size 31K, Last modified Nov-07-03)</i>` `  `

(b) HTML source code

Fig. 2.11. Example of data records that are under the same parent and are separated by
 tag

To extract data items we first need to locate data region and to find each data record. Inside each data record we extract data items. All these steps are done using tag paths. This is very convenient way of extracting data from web pages,

since in HTML tree, which represents the structure of HTML code, each tag path leads to a particular location.

The tag paths of data items in data records are also used in alignment process. We presume that data items in each data records have the same tag paths. During the extraction process data items are aligned according to their tag paths in a data record.

2.2.10. Wrapper Generation

Wrapper induction step automatically derives extraction rules, which can be later reused to extract structured data from the same template web page. Usually, these extraction rules are saved as XPath sets per data region. In case of type B data region (as seen in Figure 2.8b), we also save the XPath of visually first data record item. Figure 2.12 has an example of a data extracting wrapper. The first XPath is the address of data region, second XPath is the relative XPath of each data record and the last three XPaths are data items, which can be found in each data record.

```
"/html[1]/body[1]/form[1]/table[1]/tbody[1]/tr[2]/td[1]/div[1]"
: {
      "/div[*]" : [
         "/div[1]/div[2]/div[2]/span[2]",
         "/div[1]/div[4]/p[1]/span[1]/a[1]/span[2]",
         "/div[1]/div[2]/div[1]/span[1]",
]}
```

Fig. 2.12. An example of XPath wrapper

2.3. Conclusions of Chapter 2

1. A novel method for extracting structured web data was proposed in this Chapter. The method is called ClustVX and it is based on clustering visually similar web pages elements. It is also automatic and does not require any a priori information, such as domain ontology, data models, a set of regular expressions and etc.
2. Visual and structural features of technologically sophisticated modern web pages can be accessed by loading the pages into a modern web browser and then using browser's API to retrieve the features. During a web page loading process a web browser also executes all JavaScript code, runs any necessary asynchronous requests (AJAX), retrieves additional data, such as style sheet files

(CSS), and finally presents a rendered version of the web page. The final rendered version of a web page is then used in the structured data extraction process.

3. Since web designers optimize web sites for humans, semantically similar information, such as products' prices, titles, model numbers and etc., is presented using same style. By clustering visible web page elements according to their structural and visual similarity it is possible to detect repeating visual and structural patterns of embedded structured data records. Resulting clusters contains visually and structurally similar web pages elements that are also semantically similar. This way ClustVX exploits both visual and structural features of web pages to identify underlying repeating patterns of embedded structured data.
4. Clusters of visually and structurally similar web pages elements also contain XPath location of each element. These XPaths can be employed to generate data extracting wrapper.
5. The XPath language is suitable for generating data extracting wrappers. The final wrappers are made of a set of XPath expressions, which, when executed in a right order, can extract structured data records from HTML documents.
6. Successful implementation of the proposed method revealed that it is possible to host it on a remote server and make it accessible from internet. The method could be presented as a service with an application programming interface where input could be an URL and output – XPath wrapper and structured data in machine readable format. Furthermore, generated XPath wrappers can be reused by many data extracting methods, since the XPath language is an open standard and is widely used. Thus the proposed method can be easily integrated into other systems that extract structured data from web pages.

3

A Method for Structurally Clustering Template-Generated Web Pages

In this Chapter we propose a novel scalable method to cluster template-generated web pages according to their structural similarity. The scalability of duplicate content detection algorithms motivated us to come up with a method capable of structurally clustering template-generated web pages without the need of constant pair-wise comparison of their content. Although there are highly efficient approaches (Blanco *et al.* 2011; Hernández *et al.* 2012) to this problem, they usually rely on URL pattern recognition and are prone to unusual and unexpected formats of URLs. Contrary to them, we employ exact locations (XPaths) of inbound inner-site links to cluster same template-generated web pages. We call our method *UXClust* (derived from *URL XPath Clustering*). The proposed UXClust method exploits the observation that each unique XPath location containing a link usually points to same template-generated web pages.

The results presented in this Chapter are published in (Grigalis, Čenys 2014b).

3.1. Structural Clustering of Web Pages

Template-generated web site has a limited number of templates which are used to automatically generate new instances of web pages. Upon a web page request web server queries a database, retrieves particular data record, and embeds that information into a template, which in turn is returned to web browser. So each time browsing the same template a structurally similar web page is returned. The structural similarity of template-generated web pages is an important feature for data extraction. It is possible to write simple data extraction rules, also known as wrappers, for each kind of template, and later reuse these rules on all other web pages of the same template. This empowers extracting data from many template-generated pages using same wrapper. Consider, for example, such sites as imdb.com or ebay.com that contains thousands or even millions of web pages generated using one of their templates. By writing only one wrapper for each individual template it is possible to extract structured data from thousands of web pages (Bohannon *et al.* 2012).

Wrapper systems for data extraction are commercially popular and are the subject of extensive research over the last two decades (Blanco *et al.* 2011). Wrapper generating techniques can be broadly divided into manual and automatic approaches. Manual techniques require human annotator who given a set of example web pages marks important parts to be extracted and in such way constructs a wrapper. Automatic wrapper generating systems automatically recognize and extract structured data from a set of web pages. In both cases a set of structurally identical web pages must be present, i.e. same template-generated web pages. However, since much effort is now directed to extracting data at Web-scale (Bohannon *et al.* 2012; Cafarella *et al.* 2008; Elmeleegy *et al.* 2011; Gulhane *et al.* 2011), manually selecting or labelling same template-generated web pages becomes infeasible. The reason is simple – the Web is incredibly enormous repository of data embedded into billions of web pages. Only highly efficient and unsupervised approaches seem to be feasible to extract structured web data at Web-scale.

An equally important, but less recognized outcome of the new problem definition of Web-scale data extraction is the need to automatically organize pages of the same site into clusters, such that each cluster contains structurally similar template-generated web pages (Blanco *et al.* 2011). Then using unsupervised wrapper generating technique a high quality wrapper can be inferred for each cluster of web pages. Here again, only unsupervised and efficient method can be used to automatically identify and cluster web pages. Alternatively, if any of these two steps, i.e. wrapper generation or structural web page clustering require some human effort, then we cannot claim capability to extract data at Web-scale. Even though, as noted in (Blanco *et al.* 2011), there exist structured data extracting

techniques (Liu *et al.* 2010; Zhai, Liu 2006), that explicitly do not require such web page clustering, the latter can definitely help in organizing and synthesizing extracted data (Nguyen *et al.* 2011). Furthermore, unsupervised structural clustering of web pages can substantially improve the accuracy and recall of data extraction techniques.

As we see, unsupervised clustering of web pages is equally important problem as wrapper induction and is studied in (Blanco *et al.* 2011; Chakrabarti, Mehta 2010; Crescenzi *et al.* 2005; Joshi *et al.* 2003). However, most state-of-the-art structural web page clustering techniques are purely content based and in most cases have at least quadratic running time complexity. The high dependency on web page content analysis creates a fundamental issue: these techniques do not scale to large web sites (Blanco *et al.* 2011). Database generated web sites can have millions of web pages and there could be thousands of those web sites that we are interested in clustering in a reasonable amount of time. Even though state-of-the-art system in XML clustering, the Xproj (Aggarwal *et al.* 2007), has a linear complexity, it still requires an estimated time of more than 20 hours to cluster a site with million pages (Blanco *et al.* 2011). There is a need of less content-dependent techniques to cluster large amount of web pages.

3.2. Running Example

We now take an example to demonstrate the overall working of the proposed method. Each web page is actually a source code visually rendered by a web browser. In Figure 3.1 a set of three screenshots of visually rendered web pages from argos.co.uk is shown. Although all these three pages contain information about totally different products, i.e. sunglass, bed, and a digital camera, they all share almost identical visual appearance. This is because all three pages share common layout structure and have very similar web page source code.

The most common language in which web pages source code is written is called Hypertext Markup Language, abbreviating HTML. To better understand, modify and analyse web pages, HTML source code can be represented as a tree structure. Visually similar rendered web pages have structurally similar underlying HTML tree structure. In other words, all three pages share the same structural template and are template-generated. In this particular case, the only significant main differences between structurally similar web pages are the different text fragments describing product, such as product title, product price, product description, etc. This text usually comes from an underlying database. Sometimes there is a need to access exact parts of a web page. For example, one can be interested in extracting a particular link or text. In such cases XPath expressions are used access specified places of a given web page. XPath is a query

language for selecting specific tree nodes from a document represented as a tree structure (remember, that each web page can be seen as a tree).

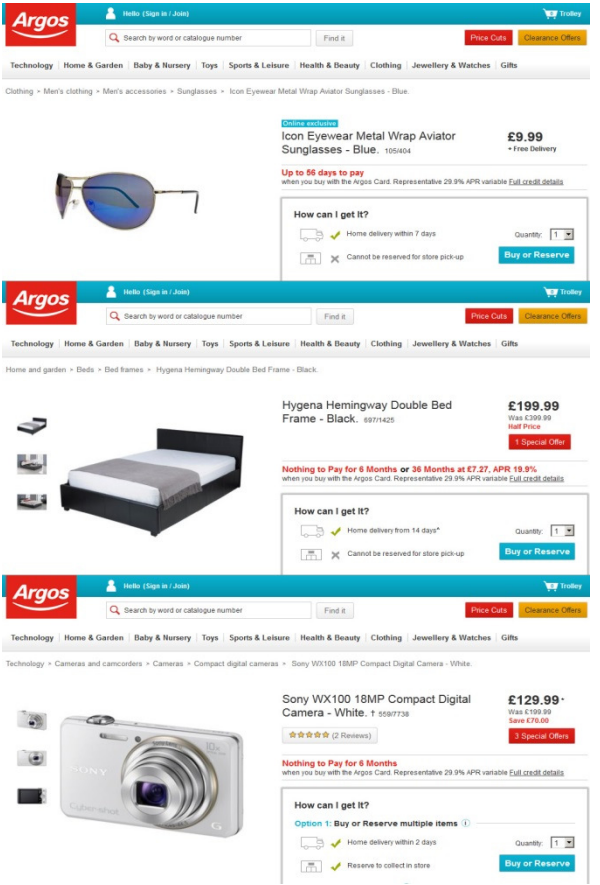


Fig. 3.1. Screenshot example of three same template-generate web pages taken from argos.co.uk

Another essential feature of hypertext documents (web pages) is that every page is linked to another. Returning to our example, every web page in Argos.co.uk web site has links to other pages in the same site. We consider only inner-site links, i.e. links that point to pages in the same domain. For example, in each of screenshots in Figure 3.1 at the top there is a menu link zone, where every link points to a particular category. Each link has its unique address (XPath expression) in a web page tree structure, for example */html/body/center/div[2]/ul/li[4]*. If we take the XPath address of a link and the

URL of the link, let's say *http://www.argos.co.uk/static/Browse/ID72,* then every link from a web page can be represented as a tuple consisting of URL and its XPath address: (*/html/body/center/div[2]/ul/li[4], http://www.argos.co.uk/static/Browse/ID72*). This observation indicates that links extracted from pages of the same template will have the same XPath. This way, we can later cluster links according to their XPath address.

Fig. 3.2. Links on the same location points to same template-generated web pages

For another example, consider Figure 3.2 where each highlighted link lead to product page. Although all the products are different, nevertheless these links point to same template-generated web pages, i.e. product pages. Furthermore, each link has the same XPath address and they are located in the same location in each page.

Of course, the Argos.co.uk web site contains many templates, for example, each for product categories pages, category homepages, comment pages, etc. The goal of our method is to group all those pages into clusters, such that each cluster contains only structurally similar pages that share the same template. To achieve this goal we exploit the available information about each link and its exact XPath location in a web page.

Web page designers optimize visual appearance of web pages for humans. There is visual style common to all templates of the same web site. Thus humans are quickly used to particular template style and can concentrate on reading the main information. This observation suggests that there are a limited number of

different templates in a same web site and there are a limited number of places in a template where inner-site links are displayed. The regularity of template style and its optimization for humans makes each unique location of link to point to the same template web page. If it was otherwise and there were no regularities, then humans would find it difficult to navigate such web site.

To sum up, our method is based on the following observations:

1. There is a limited number of different style templates in one particular template-generated web site.
2. There is a limited number of inner-site link locations in all templates of the same site.
3. Each individual location in a web page containing a link usually points to structurally similar web pages.

As a result we can determine to which template each of XPath containing a link points to. We calculate structural similarity of web pages by comparing their tree structure and binding each template to particular link location. Computationally intensive task of finding clusters of structural similar web pages is done only once per unique web link location in HTML tree. If we later encounter a link from the same location in a web page tree, we know that it points to already determined cluster. There may be thousands of unique web pages that are linked from the same location. For example, Argo.co.uk product pages have pagination links, i.e. a list of products from particular category is divided among many pages. Each such page has a limited number of places for products and each such link points to a product page. Each product page, as we know from earlier discussion and from Figure 3.1, is generated using the same template and, belongs to the same cluster of structurally similar pages. Each cluster has a set of link locations that point to it. We exploit this data to cluster web pages without analysing their content. Later we merge clusters containing structurally similar web pages by analysing only a small fraction of pages from each cluster.

3.3. Structural Similarity of Template-Generated Web Pages

Structural similarity of web pages describes how similar is their HTML source code. Since we are only interested in structural similarity of template-generated web pages the actual text that does not belong to HTML markup code is not taken into account when calculating similarity. In other words, the textual content that is visible to ordinary web site browser is the object of more importance to a related research field, i.e. duplicate or near duplicate web pages detection (Broder *et al.* 1997). So in this Section we are going to describe a few similarity measures that are used to compare structural similarity of web pages

and not their textual content. These methods and measures often rely on HTML tree and XPaths.

For notation purposes we assume P_1 and P_2 to be two HTML web pages, T_i to be a ordered tree structure of corresponding HTML code, and N_i to be a set of HTML tree nodes of which L_i are leaf nodes, and i={1,2}. Leaf nodes of HTML tree are those nodes that themselves do not have any children nodes.

As noted in the previous Section of this Chapter, each web page can be viewed as an ordered tree structure. So the most straightforward technique to calculate similarity of two web pages is to calculate the cost of transforming one tree structure into another. For this purpose basic operations like inserting, deleting, replacing or moving individual tree nodes or entire sub-trees in the tree structure are associated with certain costs to perform these operations. In case of structural trees created from HTML markup code, the problem is usually a bit more simple, since the root node is known, the sibling nodes are ordered and as the sub-trees (especially when documents are same template-generated) are hardly ever changing their distance to the root node (Gottron 2008). This can be used as a similarity metric by normalizing the number of edit operations with the number of nodes in the tree representing the larger web page (Buttler 2004). If, as noted before, P_1 and P_2 are two web pages to be compared, *editDistance*() is a function that calculates basic web page tree operations (like inserting, deleting, replacing) required to transform P_1 to P_2 and *max*() is a function returning the biggest number, then tree edit distance (TED) can be formalized into the following formula at (1):

$$TED(P_1, P_2) = \frac{editDistance(P_1,P_2)}{\max(|N_1|,|N_2|)}. \tag{3.1}$$

However, basic tree edit distance algorithms (Demaine, Mozes 2007; Nierman, Jagadish 2002; Tai 1979; Zhang, Shasha 1989) have a big drawback – they do not scale well, because they have at least a linear dependence on the size of each HTML tree and quadratic dependence on the combined size of the two trees. A faster method to compare web pages is to use the *pq-gram distance* (Augsten *et al.* 2005), which approximately match ordered labeled trees. The pq-grams of a tree are all its sub trees of a particular shape. Intuitively, two trees are close to each other if they have many pq-grams in common. For a pair of trees the pq-gram distance can be computed in O(*n* log *n*) time and O(*n*) space, where n is the number of tree nodes (Augsten *et al.* 2010). For p > 0 and q > 0, the pq-gram distance, pqGDist(T_1,T_2), between two trees T_1 and T_2 having corresponding sets of pq-grams $P^{p;q}(T_1)$ and $P^{p;q}(T_2)$,is defined as follows (Augsten *et al.* 2005):

$$pqGDist(T_1, T_2) = 1 - 2 \cdot \frac{|(P^{p;q}(T_1) \cap P^{p;q}(T_2)|}{|(P^{p;q}(T_1) \cup P^{p;q}(T_2)|}. \quad (3.2)$$

Another way to measure structural similarity between two web pages is to compare tag paths of each leaf node (Joshi *et al.* 2003). Leaf nodes are those nodes in HTML tree that do not have any children. Tag path of such node is concatenated string of tags name leading from root node to the particular leaf node. Such concatenated string can be acquired by calculating absolute XPaths of particular leaf node. See Figure 3.3, where a bag of such strings is shown.

```
/html/body/div/div/a
/html/body/div/center/table/tbody/tr/td/div/a
/html/body/div/p/div/ul/li
/html/body/div/div/p/div/ul/li
/html/body/footer/div/p
```

Fig. 3.3. A sample set of XPaths from a web page

So in a such way each web page P_i can be represented as a bag of XPath strings, formally by set $xp(P_i)$ of strings. A common paths distance (CPD) measure can be calculated via intersection of two XPath sets $xp(P_1)$ and $xp(P_2)$ generated from two web pages P_1 and P_2. See the following formula (Gottron 2008):

$$CPD(P_1, P_2) = 1 - \frac{|xp(P_1) \cap xp(P_2)|}{\max(|xp(P_1)|, |xp(P_2)|)}. \quad (3.3)$$

3.4. The Proposed Method

In this Section of the Chapter we in detail discuss our proposed approach to cluster structurally similar web pages. See Figure 3.4 where the process flow is presented of the proposed method implemented as a system. The overall working process consists of five main steps: web site crawling, link extraction, URL and XPath tuple generation, first stage approximate clustering, and final clusters refinement stage. In the following sub-sections we describe each of these steps with more details.

3.4.1. Crawling and Extracting XPath of Inner-site Links

Web site crawling step is used to collect web pages from given web sites. The process begins with providing to the method a seed URL from which it begins

crawling process. We utilize breadth-first crawling strategy, where the algorithm recursively follows collected hyperlinks. The priority to follow and download is given to first seen URLs, in other words, first-in-first-out method is used. The main result of this web site crawling step is downloaded documents that are now ready to be processed.

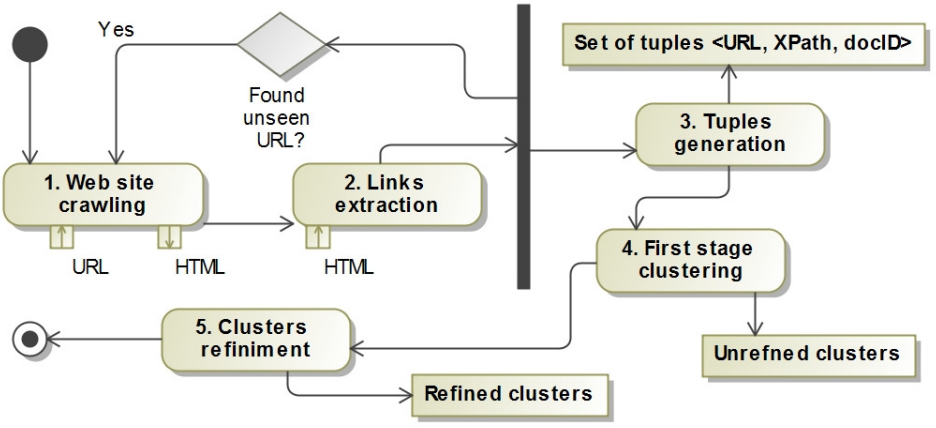
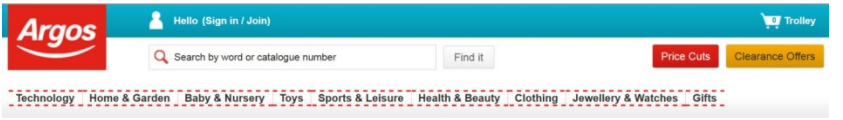


Fig. 3.4. The activity diagram of the proposed UXClust method implemented as a prototype system

Each downloaded web page is processed to create a HTML tree from the source code. This is done to simplify link extraction: we execute XPath expressions to select URLs. Of course, a more straightforward approach would be to utilize regular expressions to extract links; however, we are interested in obtaining not only URLs, but also and their location in the document tree.

3.4.2. URL and XPath Tuples Generation

Each unseen URL is forwarded to web site crawling process and saved into a tuple set consisting of extracted URL, XPath location of URL in originating document, and ID of downloaded page. For an example, see Figure 3.5(a) where a snippet of rendered web page from argos.co.uk is shown. This snippet contains a list of menu links pointing to different categories of the site, such as “Technology”, “Home & Garden“, etc. Link collection process extracts these links together with additional data describing the link: its XPath and the URL itself. Each tuple also gets and *DocID* which later lets identify downloaded page and find corresponding tuple.



(a) a snippet of rendered web page with highlighted links

LINK TEXT	URL	XPATH	DOCID
Technology	http://www.argos.co.uk/...33006169.htm	/html/body/div/div[4]/div/ul/li[1]/a	1
Home & Garden	http://www.argos.co.uk/...33005908.htm	/html/body/div/div[4]/div/ul/li[2]/a	2
Baby & Nursery	http://www.argos.co.uk/...33005732.htm	/html/body/div/div[4]/div/ul/li[3]/a	3
Toys	http://www.argos.co.uk/...33006252.htm	/html/body/div/div[4]/div/ul/li[4]/a	4
Sports & Leisure	http://www.argos.co.uk/...33006346.htm	/html/body/div/div[4]/div/ul/li[5]/a	5
Health & Beauty	http://www.argos.co.uk/...33005814.htm	/html/body/div/div[4]/div/ul/li[6]/a	6
Clothing	http://www.argos.co.uk/...33006249.htm	/html/body/div/div[4]/div/ul/li[7]/a	7
Jewellery & Watches	http://www.argos.co.uk/...33005868.htm	/html/body/div/div[4]/div/ul/li[8]/a	8
Gifts	http://www.argos.co.uk/...33005782.htm	/html/body/div/div[4]/div/ul/li[9]/a	9

(b) a sample list of generated tuples

Fig. 3.5. An example of tuple generation results

It's worth noting that during the web site crawling process we download only unseen unique links. In case a page contains several XPath locations with the same link (URL) we extract only the first one, i.e. the one located in the topmost part of the web page tree compared to the other locations. Only unique URLs are downloaded and later forwarded to the clustering process.

3.4.3. Approximate Clustering

The main task of clustering in this first stage (marked number 3 in Figure 3.4) is to group downloaded pages in a way that each group contains only pages who's URL originate in the same XPath location in HTML tree. In this stage we do not analyse the content of downloaded pages. Only associated tuples are used to cluster those pages. To be more precise, we simply group pages according to their inbound link XPath address. For example, if we have three XPaths [*/html/body/div[2]/a, /html/body/div[2]/a, /html/body/p[2]/center/a*] then we will have two resulting clusters. One would contain [*/html/body/div[2]/a, /html/body/div[2]/a*] and the other [*/html/body/p[2]/center/a*]. So any resulting cluster contains only links with identical XPaths, i.e. links extracted from the same

location. Figure 3.6 portrays a situation where three clusters are formed. Notice, that there are three unique XPath locations that are shared between many links. All these links (and pages) are clustered according to the unique XPaths and thus there are three resulting clusters.

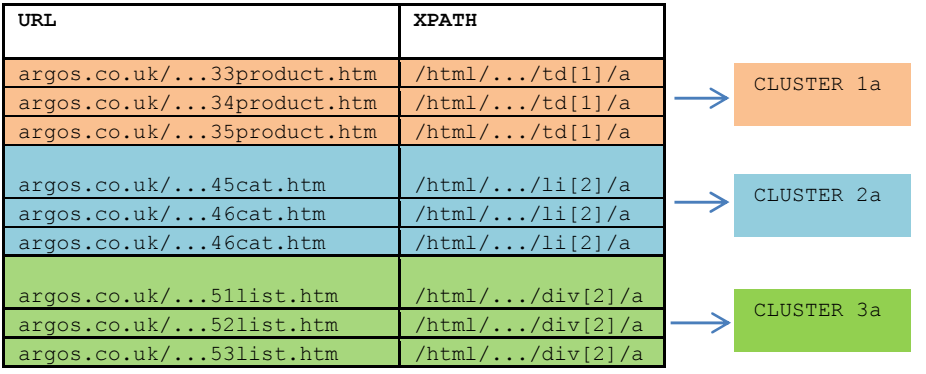


Fig. 3.6. Links clustering according to their XPaths

As it is discussed before, template-generated web pages have visual and structural regularity. Links in same template-generated web pages appear on the same locations and have same XPaths. Furthermore, each template based and database backed web site has only a limited number of different templates. This way, only a limited number of unique locations in HTML tree containing links exist. So grouping web pages by XPaths of originating URL location is enough to approximately cluster them according to their structural similarity. Another important outcome is that the computational complexity of this clustering technique is very low compared to two other baseline methods (see Chapter 4).

3.4.4. Clusters Refinement

The previous clustering process clusters web pages according to their inbound-links XPaths. Depending on the design of a web site template there could be from tens to a few hundred different XPaths with links, and web pages are grouped into the same amount of clusters. We call these clusters as *unrefined clusters*, because the clustering process takes into account only XPaths of inbound links and do not compare the structural similarity of web pages. It means that web pages belonging to separate clusters, indeed, can be structurally identical. Consider for example the list of links and their XPath locations in Figure 3.5. Although all those links have different XPath address, they all point to structurally

identical web pages. These web pages are same template-generated and display list of subcategories in any of main categories, such as Toys, Technology, Gifts, etc.

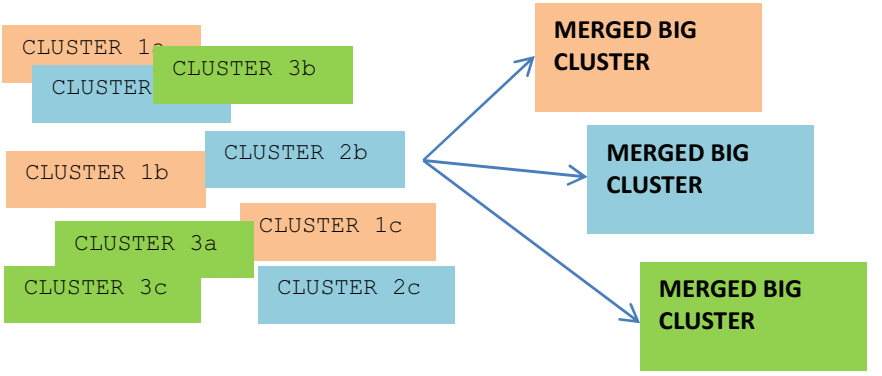


Fig. 3.7. Clusters refinement process

The basic goal of this stage is to merge any two or more unrefined clusters if they are structurally similar. For an example first consider Figure 3.6. In the first figure we see that web pages are approximately clustered according to their XPath location. We get three unrefined clusters: “CLUSTER 1a”, “CLUSTER 1b”, “CLUSTER 1c”. During a web site crawling we typically generate many clusters that may share the structural similarity (see Figure 3.5 where links with different XPaths point to same template pages). So we need to merge these clusters as portrayed in Figure 3.7.

Depending on web site template design there could be from a few hundreds to a few thousands unrefined clusters. During pilot experiments we observed that many unrefined clusters contain only a small amount of URLs. These URLs may often be found in other unrefined clusters, where thousands of unique URLs are located. These minor unrefined clusters may point to a top ten products page, news pages, policy pages, “about us” pages, etc. As it is obvious, these kinds of pages usually do not contain important structured data or that the data is redundant like in top ten products page example. So we decided to introduce *CUT-OFF threshold (*we use value of 100*)* to remove some unrefined clusters. This threshold is a limit of minimum unique URLs per cluster. If any cluster contains less than a cut-off threshold amount of links, it is removed. This way only high quality, many URL containing unrefined clusters are pushed to the next final stage.

The final refinement stage of the proposed method takes a predefined (we use value of 5) amount of sample pages from each of unrefined clusters. The content

of sampled web pages is analysed to generate a common template fingerprint of that particular cluster. The idea to use web page fingerprints instead of direct HTML tree comparison comes from closely related research field addressing duplicate content detection on the Web. The main idea there is to detect identical or near duplicate web pages by analysing their structure and content. These works (Broder *et al.* 1997; Manku *et al.* 2007) are motivated by the fact that web contains many web pages on different domains with identical or near identical content. A user searching the Web is only interested in unique content on each web page from search results list and duplicated content is no use for him. So search engines try hard to remove similar web pages from occurring in a search results. Since web pages come from thousands of domains and there could be literally millions of web pages to compare each against another a scalable technique was developed to be able to cope with such big amount of comparisons. However, we do not directly utilize web page shingling (Broder *et al.* 1997) and min-hashing (Manku *et al.* 2007) techniques to generate the fingerprints. Although they are very scalable but, on the other hand, they also are very approximating, i.e. inner-site template differences can be ignored and whole site can be seen as one template. So instead shingling and min-hashing to generate a fingerprint we employ bag of paths method (Joshi *et al.* 2003). For each sampled web page from a cluster we extract all HTML tree paths that do not contain text. Then paths occurring only above threshold value of 0.3 are taken to be included into a fingerprint. So our version of web page fingerprint is a set of selected tag paths.

Generated fingerprints for each template of clustered web pages are used to detect similarities among clusters. As discussed above, two or more clusters may actually contain same template-generated web pages. If such similarity among two clusters is detected and it is greater than predefined (we use value of 0.8) threshold value, the refinement process merges those two clusters and forms a new one. The process is repeated until no new clusters can be formed. Here basically we employ the union-find algorithm.

3.5. Conclusions of Chapter 3

1. In this Chapter a novel unsupervised method for structurally clustering template-generated web pages was proposed. The method is called UXClust and it leverages XPath locations of inbound inner-site links to speed up clustering time.
2. Some important observations have been made about template-generated web sites. Each template-generated web site typically has a limited number of different style templates. There are also a limited number of inner-site link locations in all templates of the same site.

Furthermore, each individual location with a link in template-generated web page typically points to structurally similar web pages.

3. By extracting the XPaths of inner-site links it is possible to exploit this this meta-data about linked web pages to approximately cluster them without analysing their HTML content. Computationally intensive task of generating clusters with structural similar web pages can be done only once per unique web link location in HTML tree.
4. Traditional web page structural distance calculation methods, such as Jaccard similarity coefficient of XPath sets between two web pages can be used to refine approximately clustered web pages. Although this technique is computationally expensive, however, it is possible to take only a small sample of web pages from each approximate cluster and use the calculated similarity coefficient to merge them.
5. The proposed method can be easily integrated into many data extraction systems, since any totally basic web site crawling technique can be used to download web pages for clustering. It is not important in what order or with what method web pages are download. The only requirement is to additionally to URLs also save the XPath locations of these inner-site links.

4

Experimental Evaluation of the Proposed Methods

This Chapter is dedicated to experimentally evaluating the two proposed methods. The UXClust for structurally clustering template-generated web pages and the ClustVX for extracting structured data from template-generated web pages.

Parts of this Chapter are published in (Grigalis, Čenys 2014a), (Grigalis, Čenys 2014b), (Grigalis 2013), (Grigalis *et al.* 2012b).

4.1. Evaluating the Method for Structurally Clustering Template-Generated Web Pages

In this Section of the Chapter we experimentally evaluate the proposed approach to structurally cluster template-generated web pages. We also compare our proposed approach to two baseline methods: common XPaths (Joshi *et al.* 2003) and pq-grams (Augsten *et al.* 2005). Since, to the best of our knowledge, there is no benchmark dataset suitable for our system, we need to create it. Most of datasets containing crawled web pages at best contain saved web pages and their URLs. However, our proposed approach utilizes XPath addresses of inbound

inner-site links. So in addition to physically saved web pages we also need to have those XPath addresses. And as noted before, no dataset can provide that necessary meta-data about saved web pages. Experiments were conducted on laptop computer with Ubuntu 12.04 operating system, Intel® Core™ i7-2670QM CPU @ 2.20GHz, 8 GB RAM, 7200 RPM hard drive.

4.1.1. The Dataset

To create a dataset consisting of web pages with required additional information about inner-linkage we programmed a basic web site crawler. The crawler is implemented in Python programming language. We utilize a breadth-first crawling approach where all neighbouring web pages are crawled first. This is contrary to a depth-first crawling which prioritizes exploring web site as far as possible along each linking branch before backtracking. It is demonstrated (Najork, Wiener 2001) that traversing the Web graph in breadth-first search order is a good crawling strategy, as it tends to discover high-quality pages early in the crawl. Implemented crawler does not uses cookies and is single-threaded, thus it is suitable to download one page from one site at a time. However, during the crawling process we ran multiple instances of the crawler – each for different web site. The crawling algorithm encompasses a URL cleaning procedure, which removes forced session id tokens, such as *phpsessid, cfid, aspsessionid*, and etc.

We chose 14 web sites containing structured data in template-generated web pages. In Table 4.1 data about each web site is presented. The data include web site address, total amount of downloaded pages, total size of web pages in gigabytes, average size of downloaded web page in kilobytes, total amount of unique XPath locations containing inner-site links, and total amount of unique XPath locations with cut-off threshold applied, i.e. XPath locations containing more than 100 unique inner-site links. The last row contains aggregated data among all web sites. As we can see from the table, more than one million web pages where downloaded totalling in size of 119 gigabytes on disk. Average size of single web page among all web sites is 99 kilobytes. A more complicated design of a web site results in more XPath locations where inner-site links can be found. Some sites, such as argos.co.uk and bigbox.lt, have such sophisticated design that there are round five thousands XPath locations. Each such XPath corresponds to one unrefined cluster. To simplify similarity calculation among such big amount of clusters and to reduce running time, we introduced a cut-off threshold. The cut-off threshold lets us to disregard all clusters that have less than predefined threshold limit of unique pages. In our experiments the threshold was equal to 100. In the last column of the table a number of filtered clusters are listed.

Table 4.1. Basic data about each web site used in experimental evaluation

Nr.	Web site	Pages	Total size in GB	Average Size in KB	URL XPaths	URL XPaths (*cut-off*)
1.	argos.co.uk	111142	7.33	69.20	4472	224
2.	azon.lt	102313	18.28	187.32	306	86
3.	bigbox.lt	120557	27.03	235.12	5055	145
4.	citylights.com	13698	0.34	26.29	553	19
5.	currys.co.uk	9993	0.67	70.42	1078	9
6.	elshop.lt	90001	2.27	26.44	246	36
7.	ikea.com	122686	10.16	86.83	1754	117
8.	ilterzogirone.it	66404	3.64	57.55	1317	109
9.	imk.lt	74295	14.46	204.14	1864	116
10.	iristorante.it	116410	5.35	48.21	1196	119
11.	kompiutera.lt	21623	1.08	52.24	368	35
12.	smartbuy.lt	15638	0.41	27.62	441	29
13.	tesco.com	88773	15.64	184.76	3618	192
14.	varle.lt	117514	12.44	110.96	875	61
Average:		76503	8.51	99.08	1653	92
Total:		1071047	119.1	–		

All data about each downloaded web page is stored in a relational MySQL database. One table is dedicated to store all data: URL, XPath, DocID, and web site ID. Downloaded web pages are stored in file folders. There is one folder for each web site. The file names of stored web pages are identical to DocIDs, which are stored in a database table. Running time for structural clustering of saved web pages includes database queries to retrieve and save data.

4.1.2. Ground Truth and Measurements

For each web site we identified one kind of template with most interest to us. In many cases these are template-generated pages containing product data, such as title, price, picture, description, etc. Then we manually analyse URLs of these pages to identify repeating patterns, such as keywords or URL structure. A regular

expression is written for each web site to match those URLs with high interest to us. This way we generate a ground truth (golden data) that is used to calculate precision, recall, of our proposed clustering method. The precision of a given cluster is the fraction of web pages in its computed cluster that are also found in the corresponding ground truth cluster. The recall of a given cluster is the fraction of web pages from the corresponding ground truth cluster that were grouped into the same computed cluster. To calculate these measures, that are taken from information retrieval research field, we first need to describe all possible outcomes of clustering process. Our main goal is to assign two or more web pages to the same cluster if and only if they are similar. A true positive (TP) decision assigns two structurally similar web pages to the same cluster; a true negative (TN) decision assigns two structurally dissimilar web pages to different clusters. There are two types of errors that can occur. A false positive (FP) decision assigns two structurally dissimilar web pages to the same cluster. A false negative (FN) decision assigns two structurally similar web pages to different clusters. Then the Precision and Recall is calculated as follows:

$$Precision = \frac{TP}{TP+FP} \ , \tag{4.1}$$

$$Recall = \frac{TP}{TP+FN} \ . \tag{4.2}$$

4.1.3. Selecting Parameters for Two Baseline Algorithms

Since pg-Grams and Common Paths (CP) methods are very computationally expensive and result in a long running time (for detailed comparison of running times please see Table 4.3), we decided to take a much smaller amount of web pages from each web site to benchmark them. First, we test the two baseline algorithms with the first web site from the dataset, namely with *argos.co.uk*. We check if the different number of selected web pages has any significant impact on precision and recall. As seen in Figure 4.1 and Figure 4.2, there is no significant difference on precision and recall with both algorithms if we select a different number of web pages. However, there is a significant difference in terms of running time: the more pages are selected to cluster, the longer the running time. Actually, the running time increases almost exponentially with both algorithms.

Thus for pg-Grams we take only 100 web pages from each web site and use p=2 and q=3 values as suggested in original paper (Augsten *et al.* 2005). Similarly, for a faster Common Paths method we take 1000 web pages. The amount of taken

web pages is also indicated in parentheses next to the algorithm name in Table 4.2 and Table 4.3. For our UXClust method we take all web pages as seen in Table 1. This way, of course, the size of ground truth clusters differs among three benchmarked methods. The *GT-P* columns under each method in Table 4.2 indicate the number of ground truth pages for each web site. For each of three tested methods we use the same 0.8 web page similarity threshold value.

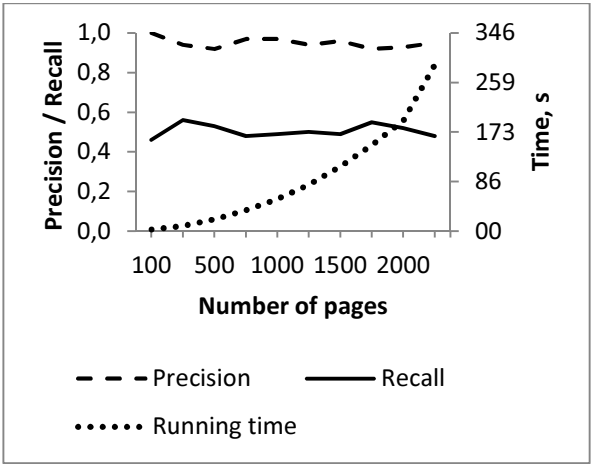


Fig. 4.1. The effect on recall, precision and running time by selecting different numbers of pages for Common Paths algorithm

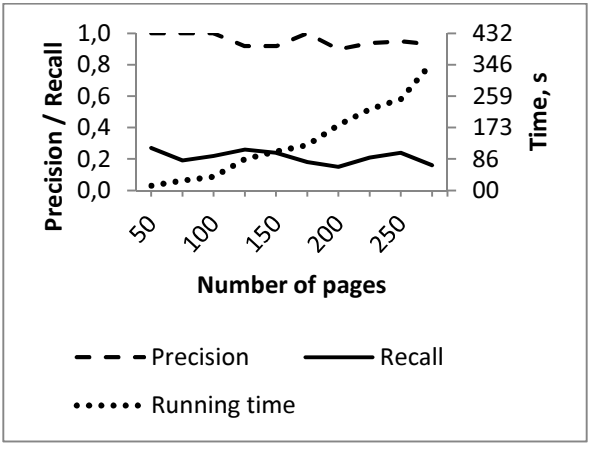


Fig. 4.2. The effect on recall, precision and running time by selecting different numbers of pages for pg-Grams algorithm

4.1.4. Results

Table 4.2. The results of using the proposed and two baseline algorithms to cluster structurally similar web pages

Nr.	Web site	UXClust			pg-Grams (100)			CP (1000)		
		GT-P	*P*	*R*	*GT-P*	*P*	*R*	*GT-P*	*P*	*R*
1.	argos.co.uk	30460	0.92	0.72	40	1	0.15	332	0.94	0.49
2.	azon.lt	36643	1	1	34	1	0.32	370	1	0.87
3.	bigbox.lt	10282	0.68	0.94	13	0.88	0.54	176	0.8	0.96
4.	citylights.com	2026	1	0.64	22	0.33	0.36	190	1	0.56
5.	currys.co.uk	2009	1	0.81	56	1	0.3	627	1	0.45
6.	elshop.lt	22019	1	1	22	1	0.55	244	1	0.76
7.	ikea.com	4909	1	1	9	1	0.56	99	1	1
8.	ilterzogirone.it	2866	1	1	10	1	0.7	49	1	0.98
9.	imk.lt	10173	0.96	1	16	0.82	0.56	191	0.95	0.93
10.	iristorante.it	967	1	0.9	4	1	1	35	1	0.74
11.	kompiutera.lt	11440	0.54	1	41	0.44	1	528	0.53	1
12.	smartbuy.lt	3632	1	1	36	1	0.25	396	1	0.99
13.	tesco.com	13066	0.97	0.96	30	0.81	0.83	274	0.79	0.15
14.	varle.lt	59450	1	1	48	1	0.73	483	0.98	0.98
	Average:	14996	0.93	0.93	27	0.88	0.56	285	0.93	0.78

Table 4.2 lists the precision (P) and recall (R) of applying our proposed (UXClust) and two baseline (pg-Grams and Common Paths) clustering algorithms on each web site. As seen in the table, our proposed UXClust method achieves 0.93 average precision and recall and outperforms two other baseline methods. The pg-Grams algorithm performs worst. We believe it is possible to tune up this algorithm to achieve better results with structurally sophisticated web pages, however, it is beyond the reach of this thesis. Common Paths and our UXClust approaches achieve the same precision as in both of them the same Jaccard similarity measure over the XPaths of web pages is used to determine structurally similar templates. However the use of XPath clustering in UXClust significantly increases recall from 0.78 to 0.93.

Table 4.3. Running time comparison between proposed and two baseline methods

Nr.	Web site	UXClust		pg-Grams (100)		CP (1000)	
		Time (s)	*GT-P/s*	*Time (s)*	*GT-P/s*	*Time (s)*	*GT-P/s*
1.	argos.co.uk	31	982	49	0.82	61	5.44
2.	azon.lt	12	3053	66	0.52	71	5.21
3.	bigbox.lt	35	293	123	0.11	190	0.93
4.	citylights.com	1	2026	21	1.05	26	7.31
5.	currys.co.uk	1	2009	61	0.92	67	9.36
6.	elshop.lt	5	4403	16	1.38	37	6.59
7.	ikea.com	18	272	65	0.14	89	1.11
8.	ilterzogirone.it	12	238	70	0.14	38	1.29
9.	imk.lt	25	406	108	0.15	153	1.25
10.	iristorante.it	13	74	50	0.08	46	0.76
11.	kompiutera.lt	5	2288	54	0.76	100	5.28
12.	smartbuy.lt	2	1816	21	1.71	31	12.77
13.	tesco.com	56	233	165	0.18	229	1.20
14.	varle.lt	17	3497	83	0.58	132	3.66
Total:		235	–	950	–	1271	–
Average:		–	893	–	0.40	–	3.14

As we further look to the results of proposed UXClust approach we see that on almost all web sites it achieves very high recall, except for the site number 4 (citylights.com). Here the recall result is just 0.64. A closer look relieved, that this happens due to unexpected behaviour of citylights.com template engine. Each time a requested URL resource does not contain corresponding entry in underlying database, the Web server returns a “sorry” message (as a small inclusion) and displays a regular web page with list of books. The template of this page is different from the one that should be returned if book data would be found. The only way to be sure of returned template is to verify each response. However, parsing each downloaded page takes a lot of time, and our proposed method tries to avoid it. That way, the clustering recall of the citylights.com web pages cannot be further improved using our proposed method. We consider this site to be an exception.

The precision of UXClust drops when web pages from different templates are clustered together. This happens on 6 out of all 14 web sites. The main difficulty here is to determine the similarity of two web pages. Our experiments revealed that the bag of XPaths approach (Common Paths), which we use in our algorithm, is not always working on modern web pages. In some sites, there are very small differences between different templates. Furthermore, a template of a product page may differ a little depending on the product type. In those cases only a human viewer could tell if those two pages are similar and the similarity may be more semantic one, than a visually structural.

In Table 4.3 we compare the running time of our proposed and two baseline methods. Recall that each method in run with different amount of web pages. Thus we calculate two scores: the whole running time in seconds and normalized measure of ground truth pages clustering speed per second (GT-P/s). The latter reveals how quickly ground truth cluster is formed. As we see in Table 4.3, our proposed method outperforms two other approaches by order of magnitude: UXClust is able to cluster 893 ground truth pages per second, while Common Paths method clusters just 3.14 ground truth pages per second and pg-Grams – just 0.40. Proposed UXClust approach is able to structurally cluster more than a million web pages in just 235 seconds.

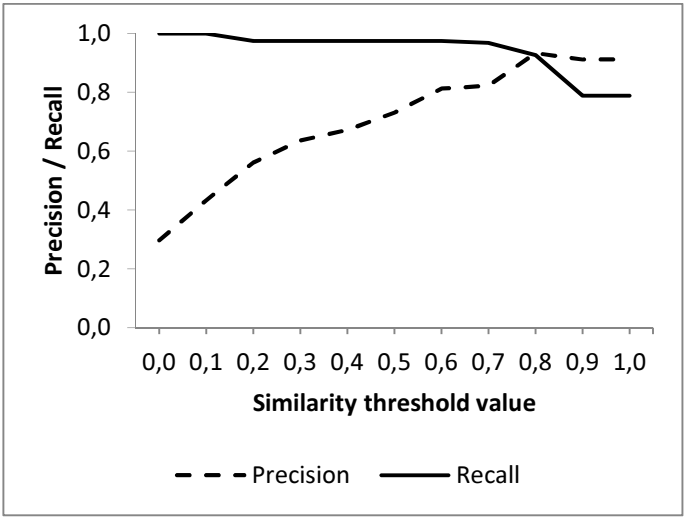


Fig. 4.3. Similarity threshold parameter effect on average precision and recall

Thresholds Impact on Overall Precision, Recall and Running Time

We also investigate the trade-offs between precision, recall and running time. To do that we vary values of all four thresholds used in proposed UXClust method, i.e. the web page similarity threshold, XPath occurrence threshold, XPath sampling threshold and CUT-OFF threshold (for a detailed explanation of each threshold please refer to the Chapter 3). We show the running time curve only with those thresholds that have considerable effect on running time.

In Figure 4.3 we see what happens to average precision and recall on all web sites when web page similarity threshold increases from 0 to 1. As we see, precision starts from around 0.3 and reaches its maximum at around 0.9, while recall starts from 1 and drops to 0.8. The precision and recall intersects when similarity threshold is around 0.8. This is the exactly same value of web page similarity threshold that is used in our proposed algorithms.

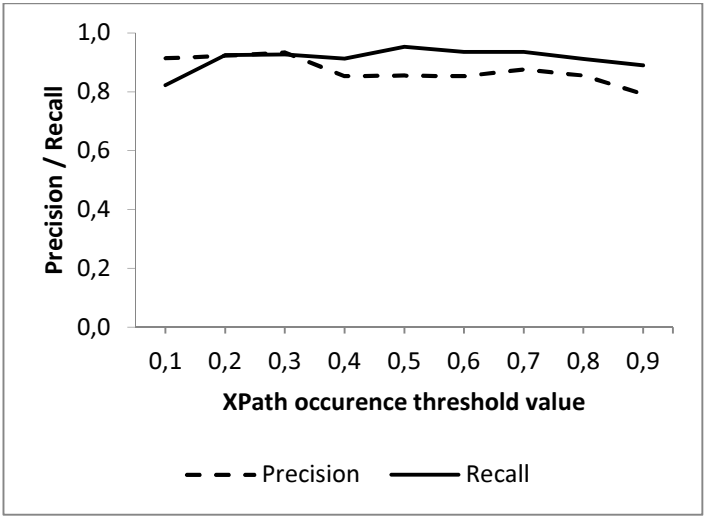


Fig. 4.4. XPath occurrence threshold parameter effect on average precision and recall

While conducting experiments we also noticed that precision and recall for each web site may differ a few percent on each new run of our algorithms. This happens because web page sampling algorithm uses random function to select a few samples of web pages to calculate their fingerprint. Depending on sampled web pages the resulting fingerprint may differ a bit. This leads to different similarity calculation results in clusters refinement stage. However, those

differences on precision and recall are no more than a few percent. More over the similarity threshold has no effect on running time.

As shown in Figure 4.4 the best results in terms of precision and recall are achieved when setting the XPath occurrence threshold between 0.2 and 0.3.

Varying XPath occurrence threshold value has small effect on precision and recall. These results mean that in clusters fingerprint generation process there is a small difference between selecting all found unique XPaths from sampled web pages or including just most common ones. This threshold also does not have any effect on running time, thus we do not include running time axis.

As seen in Figure 4.5 the more web pages are taken into sampling (higher threshold value) the longer UXClust method runs. Actually, we can see exponential growth in running time as XPath sampling threshold parameter increases. While selecting appropriate value for the threshold the results of recall and precision must be taken into considerations. Here we see the best fit at 3 where running time is still low and precision and recall are above 0.9.

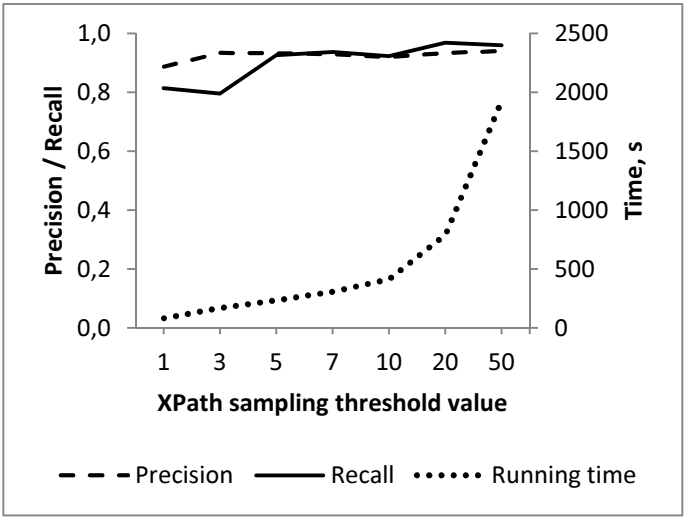


Fig. 4.5. XPath sampling threshold parameter effect on precision, recall and running time

CUT-OFF threshold value determines the smallest size of XPath clusters that are taken into web page structural clustering process. Naturally, as seen in Figure 4.6, the bigger the threshold, the less XPath clusters are compared to be merged, the less the running time. However, if we would select enormously high value of this threshold – depending on the web site, there could be no clusters left to merge and thus large part of web pages would be discarded from clustering. Our

observation demonstrates that CUT-OFF threshold value of 100 lets to achieve good results.

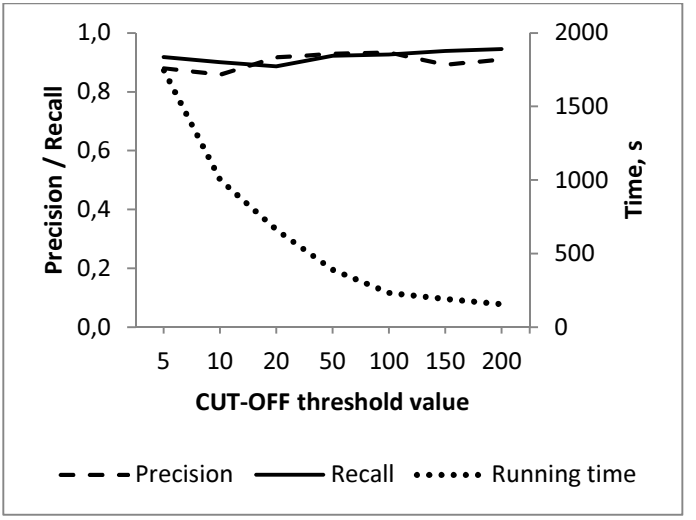


Fig. 4.6. CUT-OFF threshold parameter effect on precision, recall and running time

4.2. Evaluating the Method for Extracting Structured Data from Template-Generated Web Pages

This Section describes the extensive experimental evaluation of our proposed ClustVX method with four benchmark datasets containing more than seven thousand structured data records embedded into template-generated web pages. Given a web page from a dataset we process it with data extraction system and measure the extraction results.

4.2.1. The Datasets

Here are the four benchmark datasets that we use to evaluate structured web data extraction systems:

1. ClustVX.
2. ViNTs-2 (Zhao *et al.* 2005).
3. Alvarez (Álvarez *et al.* 2008).
4. TBDW Ver. 1.02 (Yamada, Craswell 2004).

These datasets contain search results of the Deep web, that is, pages generated using templates filled with data coming from databases. Usually these databases are accessed when user submits a web form (for example, to find all books that in title contain word „data"), and web server retrieves requested data and generates a web page. See Table 4.4 for detailed characteristics of these datasets. The total number of data records is calculated by analysing only one page per web site. In case there are many web pages per site in a dataset, we take only the first one from alphabetic. If first page contains no data records, then the second page is taken.

Table 4.4. Benchmark datasets containing template-generated web pages

Dataset:	ClustVX	ViNTs-2	Alvarez	TBDW
Sites	10	102	200	51
Pages per site	3	11	1	5
Average records per page	22	24	18	21
Total records (1st page per site)	218	2489	3557	1052

The experimental results are compared with state-of-the-art automatic structured data extraction systems called G-STM (Jindal, Bing 2010), DEPTA (Zhai, Liu 2006), FiVaTech (Kayed, Chang 2010), MDR (Liu *et al.* 2003), ViNTs (Zhao *et al.* 2005), CTVS (Su *et al.* 2011), DeLa (Wang, Lochovsky 2003), and the method proposed in (Álvarez *et al.* 2008). Some of these systems are not publicly available to download. In such case we use the reported evaluation results from the original publications.

Since some of the web pages have malformed HTML source codes, as it was done in RoadRunner (Crescenzi 2001) and DEPTA (Zhai, Liu 2006) experiments, we use the Tidy program (Paehl 2012) to clean the source code of malformed pages.

4.2.2. Evaluation Metrics

We use the precision, recall and f-score measures (which are widely used to evaluate information retrieval system) to evaluate the effectiveness of our system for extracting structured web data. For web data extraction, the recall and precision are computed based on the total number of correctly extracted data records ($DR_{correctly}$), number of extracted data records ($DR_{extracted}$) and number of actual data records ($DR_{actural}$) present in a given web pages. The F-score is based on both precision and recall. See the following formulas for calculating precision, recall and f-score:

$$Precision = \frac{|DR_{correctly} \cap DR_{extracted}|}{|DR_{extracted}|}, \quad (4.3)$$

$$Recall = \frac{|DR_{extracted} \cap DR_{actual}|}{|DR_{actual}|}, \quad (4.4)$$

$$F - score = \frac{2 * Precison * Recall}{Precision + Recall}. \quad (4.5)$$

As in FIVATECH (Kayed, Chang 2010) experimentation, we consider data record to be successfully extracted if more that 60% of its data items are extracted correctly and each of data records in a data region is correctly segmented.

4.2.3. Results with ClustVX Dataset

The first ClustVX dataset is compiled by us (see Table 4.5 for details). We chose ten technologically sophisticated modern web sites, where much of content is dynamically modified with JavaScript code and style is set by Cascading Style Sheets (CSS). Since many automatic web data extraction systems take as an input a raw HTML file they would fail to get the final HTML of a modern web page where the execution of JavaScript code and live HTML modifications are done while rendering a web page in a browser. In contrast to raw HTML handling the proposed ClustVX systems uses a modern web browser to fully render a HTML file and take the final version of the source code. To conduct a proper comparison between those systems that do not render HTML files and the proposed ClustVX system we decided to: a) fully render in the browser each web site from the dataset; b) save the resulting rendered version of the HTML source code; c) provide the rendered version of the HTML code as an input for automatic structured data extraction systems.

We compare the effectiveness of our proposed ClustVX system to three publically available to download or access automatic web data extraction systems called MDR (Liu *et al.* 2003), ViNTs (Zhao *et al.* 2005), and FiVaTech (Kayed, Chang 2010). The results in terms of absolute numbers are available in Table 4.5. We list the total number of data records available to extract in each web page. The total numbers of actually extracted Data records by each system are marked as "total". Next to it we also list the True Positive (TP) extraction results, i.e. the number of correctly extracted data records by each system. The Table 4.6 lists the results on the same dataset presented in precision, recall and f-score.

The results on this dataset reveal that out proposed ClustVX system achieves perfect precision and nearly perfect recall, 100.0% and 99.5% respectively. These

results are better than those achieved with other methods. The lowest results in terms of f-score are achieved by MDR systems. We believe it is because the MDR system is the oldest system of the all tested. During the creation of MDR the web pages were technologically simpler and the system seems to be unable to cope effectively with modern web pages.

Table 4.5. Experimental results on ClustVX dataset in absolute numbers

Nr.	Web site	TRUE TOTAL	ViNTs		FiVaTech		MDR		ClustVX	
			total	*TP*	*total*	*TP*	*total*	*TP*	*total*	*TP*
1.	currys.co.uk	20	19	19	20	20	0	0	20	20
2.	google.com	10	0	0	7	7	0	0	10	10
3.	argos.co.uk	50	50	50	3	3	1	0	50	50
4.	bestbuy.com	15	0	0	15	15	0	0	15	15
5.	clothingattesco .com	20	3	0	20	20	0	0	20	20
6.	samsung.com	15	9	0	3	3	0	0	15	15
7.	adidas.com	24	24	24	24	24	0	0	24	24
8.	amazon.com	24	0	0	24	24	0	0	24	24
9.	barnesandnoble .com	30	30	30	0	0	8	0	29	29
10.	bing.com	10	8	0	13	10	11	10	10	10
	Total:	**218**	**143**	**123**	**129**	**126**	**20**	**10**	**217**	**217**

Table 4.6. Experimental results on ClustVX dataset in precision, recall and f-score

System / Measure	ViNTs	FiVaTech	MDR	ClustVX
Precision	86.0%	97.7%	50.0%	100.0%
Recall	65.6%	59.2%	9.2%	99.5%
F-score	74.4%	73.7%	15.5%	99.8%

4.2.4. Results with ViNTs-2 Dataset

The structured data records extraction results on VINTS-2 dataset reveal, as we see in Table 4.7, that ClustVX system again achieves very high precision and recall, 98.6 % and 98.5% respectively. These results are better than those reported with G-STM (Jindal, Bing 2010) and DEPTA (Zhai, Liu 2006) systems. The main difficulties which hindered data extraction for ClustVX system are related to malformed HTML source code, which cannot be fixed even with Tidy program. Another difficulty was structurally complex lists of data records. Errors mainly occur when ClustVX system cannot correctly segment data records. Incorrect segmentation leads to extraction problems, such as when not all data items per data record are extracted, or data items belonging to the same data record are assigned into many different data records.

Table 4.7. Experimental Results on VINTS-2 dataset

System / Measure	**ClustVX**	**G-STM**	**DEPTA**
Reported actual records	2489	N/A	N/A
Extracted records	2452	N/A	N/A
Correctly extracted records	2417	N/A	N/A
Precision	98.6%	98.5%	95.1%
Recall	98.5%	96.7%	83.9%
F-score	98.5%	97.6%	89.1%

4.2.5. Results with Alvarez Dataset

Here, in Table 4.8, are the results with (Álvarez *et al.* 2008) dataset. This dataset in terms of data records number and different style web sites is the biggest of all three. The extraction results of ClustVX system are compared to the reported results of the system proposed by Alvarez et al. in (Álvarez *et al.* 2008). Same problems as we encountered with VINTS-2 dataset are present here also. Particularly, the incorrect segmentation of structured data records leads to extraction errors. However, even with these problems our ClustVX system achieves better results than the method proposed by M. Alvarez et al. We achieve 98.2% precision and 99.7% recall, while (Álvarez *et al.* 2008) achieve 97.9% and 98.3% respectively. 4.2.6. Results with TBDW Dataset

Here we compare effectiveness of our proposed ClustVX system to other state-of-the-art structured extraction systems called G-STM (Jindal, Bing 2010), DEPTA (Zhai, Liu 2006), FiVaTech (Kayed, Chang 2010), CTVS (Su *et al.*

2011), DeLa (Wang, Lochovsky 2003). With due respect to previous works and their authors, some inaccuracies have been observed in the reported results of other techniques. For example, some authors report smaller number of records in same publically available TBDW benchmark data set. For this reason we have included reported numbers where they were available. As we see in Table 4.9, the authors of FiVaTech, CTVS and DeLa report almost twice smaller number of available (actual) records in TDBW data set. This is very strange, because our thorough calculation reveals that there are 1052 records. To calculate the total amount of actual records we take first page from each of 51 sites in TBDW data set. If there are no records available in the first page (this occurred on first pages of 7th and 11th sites from TBDW), we take the second one. The same calculation method reportedly is used and in the papers of FiVaTech, CTVS and DeLa.

Table 4.8. Experimental Results on Alvarez (Álvarez *et al.* 2008) dataset

System:	**ClustVX**	**Alvarez et al.**
Reported actual records	3557	3557
Extracted records	3546	3570
Correctly extracted records	3482	3496
Precision	98.2%	97.9%
Recall	99.7%	98.3%
F-Score	98.9%	98.1%

Table 4.9. Experimental Results on TBDW data set

System:	**ClustVX**	**G-STM**	**DEPTA**	**FiVaTech**	**CTVS**	**DeLa**
Reported actual records	1052	N/A	N/A	693	693	693
Extracted records	1047	N/A	N/A	690	688	655
Correctly extracted records	1045	N/A	N/A	672	680	616
Precision	99.8%	99.8%	99.5%	97.0%	98.8%	88.8%
Recall	99.5%	96.6%	85.3%	97.4%	98.1%	94.0%

The results on TBDW dataset reveal that out proposed ClustVX system achieves nearly perfect precision and recall, 99.8% and 99.5% respectively. These results are better than those achieved with other author's methods. The precision of ClustVX and G-STM systems is the same, however our approach has much better recall and this leads to better F-score. ClustVX system could not correctly extract just one page from this data set. The main problems with precision are there were table headers of structured data records list is included as valid record.

4.3. Conclusions of Chapter 4

1. In this Chapter we have experimentally evaluated the both proposed methods: ClustVX and UXClust. The results revealed that the proposed methods consistently outperform many other state-of-the-art techniques on two tasks: structured web data extraction and web page structural clustering. For example, using the UXClust method more than one million of web pages can be clustered in less than 4 minutes. In addition to speed efficiency, the proposed method achieves higher than 90% precision and recall on all tested web sites and outperforms by a high margin two other baseline techniques. Furthermore, the proposed second method ClustVX successfully was applied to extracting structured data from many web sites containing more than seven thousand structured data records embedded into template-generated web pages. Here again, our proposed method consistently outperformed other state-of-the-art approaches in terms of precision and recall.
2. While preparing for the experimental evaluation we ran into the problem of obtaining a representative dataset. In the case of evaluating UXClust for structural clustering of web pages no dataset have been found. Similar situation was encountered and evaluating ClustVX method for extracting structured data: only a few outdated datasets have been identified. For the both experiments we have manually compiled additional benchmark datasets.
3. Although there are many approaches for extracting structured data and structural clustering of web pages, not all of them are implemented as working prototypes and available freely to download. In majority of the case the absence of freely available prototypes it was difficult to compare our proposed methods to other state-of-the-art techniques. In some of the cases we have used the reported results from original publications and could not re-evaluated those approaches with our own benchmark datasets.

4. There were also difficulties with other structured data extraction state-of-the-art approaches while reusing their reported evaluation results on TBDW public benchmark dataset. We found that many authors report a smaller amount of actually available data records to extract. We think that some kind of miscalculations occurred in the corresponding publications. Thus more publicly available and up-to-date datasets with detailed specifications are needed to further compare many approaches one against another.
5. The majority of cases in which ClustVX method incorrectly extracted data occurred while dealing with malformed HTML source code, extreme similarity (visual and structural) between data records and non-records, and incorrect segmentation of data regions. Further research is needed to better understand the issues and improve the recall and precision of the ClustVX method.
6. Returning to the UXClust method and the task of structural clustering of web pages a few open challenges also remain to be solved:
    a. Web page similarity measurement is a bottleneck of precision and recall in our algorithms. Experimental evaluation of our proposed algorithms revealed that current web page structural similarity measures find it difficult dealing with contemporary web pages, which size, in fact, can reach >200 KB of HTML code. And the real differences between two same site templates can be traced to just a few lines of code. We stipulate that there is need to include semantic analysis step to better understand the purpose of each template in a site. This could help to identify key locations or text strings that discern one template from another. Only then it will be possible to further increase precision and recall of structural clustering process.
    b. Since our approach employs XPath locations of URLs, there is need to extend current web crawlers to save this kind of additional information about downloaded web pages. We could not identify any publicly available web crawler with such functionality.
    c. There is a need to come up with methods dealing with unexpected web server behaviour. For example, we ran into a problem when some products are somehow “turned off” or not found in a database but their URL remains valid. Then web server quietly redirects request to web pages of different template, such as category list. So some kind of procedures detecting those unexpected behaviours should be employed.

General Conclusions

1. The literature review revealed that human effort requiring structured web data extraction methods bring considerable costs to organizations and are not suitable for extracting data at Web-Scale, i.e. from thousands of websites that are visually and structurally different. State-of-the-art automatic data extraction methods typically search for repeating patterns in template-generated web pages and exploit them to detect and extract embedded structured data. However, fully automatic Web-Scale structured web data extraction, at the level of precision and recall high enough to power real applications, is still a long sought goal in the data extraction community.
2. Template-generated web pages display structural and visual regularity. The proposed clustering technique can be used to detect repeating patterns of embedded structured data records in these pages. The technique is based on clustering visually and structurally similar web page elements. The resulting clusters with XPaths of web page elements can be leveraged to automatically generate data extracting wrappers.
3. It is demonstrated, that the proposed method for extracting structured data records from template-generated web page is more effective in

terms of precision and recall than all other state-of-the-art techniques used in experimental evaluation. With all four benchmark data sets the proposed method achieves higher than 98% precision and recall. The method can also process technologically sophisticated modern web pages, since web pages rendering occurs in a modern web browser.

4. XPaths of inbound inner-site links can be leveraged to significantly speed up structural clustering process of template-generated web pages. Furthermore, during the clustering process a traditional web page similarity measures such as Jaccard similarity coefficient between finite sets of XPaths can be used.
5. The proposed method for structural web page clustering is computationally superior to other state-of-the-art approaches used in experimental evaluation. It can cluster more than 1 million web pages in less than 4 minutes and in turn achieving higher than 90% precision and recall. The proposed method is more than 200 times faster than the two other traditional web page clustering methods.
6. Both proposed methods for structured web data extraction and structural clustering of template-generated web pages are completely unsupervised, automatic and domain-independent and thus they can be integrated into Web-scale data extraction systems.

References

Adelberg, B. 1998. Nodose: A tool for Semi-Automatically Extracting Structured and Semistructured Data from Text Documents*, SIGMOD Records* 27(2): 283–294.

Aggarwal, C., Ta, N., Wang, J. 2007. Xproj: A Framework for Projected Structural Clustering of Xml Documents*, In Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York: ACM Press, 46–55.

Álvarez, M., Pan, A., Raposo, J., Bellas, F., Cacheda, F. 2008. Extracting Lists of Data Records from Semi-Structured Web Pages*, Data & Knowledge Engineering* 64(2): 491–509.

Arocena, G.O., Mendelzon, A.O. 1998. Weboql: Restructuring Documents, Databases, and Webs*, In Proceedings of the Fourteenth International Conference on Data Engineering*. Washington: IEEE Computer Society, 24–33.

Augsten, N., Böhlen, M., Gamper, J. 2005. Approximate Matching of Hierarchical Data Using Pq-Grams*, In Proceedings of the 31st International Conference on Very Large Data Bases*. San Francisco: Morgan Kaufmann Publishers inc., 301–312.

Augsten, N., Böhlen, M., Gamper, J. 2010. the Pq-Gram Distance Between Ordered Labeled Trees*, ACM Transactions on Database Systems* 31(1): 1–35.

Baumgartner, R., Flesca, S. 2001. Visual Web Information Extraction with Lixto*, Very Large Databases* 1: 119–128.

Baumgartner, R., Gatterbauer, W., Gottlob, G. 2009a. Web Data Extraction System, *Encyclopedia of Database Systems.* New York: Springer, 3465–3471. ISBN 9780387355443.

Baumgartner, R., Gottlob, G., Herzog, M. 2009b. Scalable Web Data Extraction for online Market intelligence, *Proceedings of the VLDB Endowment* 2(1): 1512–1523.

Beach, T. 2013. the CSS Box Model. [Online]. [Cited 11 March 2014]. Available from Internet: <Http://Www.Unm.Edu/~Tbeach/IT145/Week08/Index.Html>.

Bergman, M.K. 2001. the Deep Web: Surfacing Hidden Value, *Journal of Electronic Publishing* 7(1): 1–17.

Berners-Lee, T. 2000. Weaving the Web : the Original Design and Ultimate Destiny of the World Wide Web By Its inventor. New York: Harper Business, ISBN 0062515861.

Bing, L. 2012. Web Data Mining. New York: Springer, ISBN 9783642194597.

Blanco, L., Dalvi, N., Machanavajjhala, A. 2011. Highly Efficient Algorithms for Structural Clustering of Large Websites, *In Proceedings of the World Wide Web Conference*. New York: ACM Press, 437–466.

Bohannon, P., Dalvi, N., Filmus, Y. 2012. Automatic Web-Scale Information Extraction, *In Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. New York: ACM Press, 609–612.

Britain, G., Hsu, C., Dungs, M., Science, I., Science, C. 1998. Generating Finite-State Transducers for Semi-Structured Data Extraction from the Web, *Information Systems* 23(8): 521–538.

Broder, A.Z., Glassman, S.C., Manasse, M.S., Zweig, G. 1997. Syntactic Clustering of the Web, *Computer Networks and ISDN Systems* 29(8-13): 1157–1166.

Buttler, D. 2004. A Short Survey of Document Structure Similarity Algorithms, *In Proceedings of the 5th International Conference on Internet Computing*. New York: ACM Press, 3–9.

Cafarella, M.J., Halevy, A. 2009. Data integration for the Relational Web, *Proceedings of the VLDB Endowment* 2(1): 1090–1101.

Cafarella, M.J., Halevy, A., Madhavan, J. 2011. Structured Data on the Web, *Communications of the ACM* 54(2): 72–79.

Cafarella, M.J., Halevy, A., Wang, Z.D., Wu, E. 2008. Webtables : Exploring the Power of Tables on the Web, *In Proceedings of the International Conference on Very Large Data Bases (VLDB)*. San Francisco: Morgan Kaufmann Publishers inc., 538–549.

Cai, D., Yu, S., Wen, J. 2003. VIPS : A Vision-Based Page Segmentation Algorithm, *Technical Report, Microsoft* MSR-TR-200.

Califf, E., Mooney, J. 1999. Relational Learning of Pattern - Match Rules for Information Extraction, *In Proceedings of the Sixteenth National Conference on Artificial intelligence*. Cambridge: MIT Press, 328–334.

Chakrabarti, D., Mehta, R. 2010. the Paths More Taken: Matching DOM Trees to Search Logs for Accurate Webpage Clustering, *In Proceedings of the World Wide Web Conference*. New York: ACM Press, 211–220.

Chakrabarti, S., Van Den Berg, M., Dom, B. 1999. Focused Crawling: A New Approach to topic-Specific Web Resource Discovery, *Computer Networks* 31(11-16): 1623–1640.

Chang, C. 2001. IEPAD : Information Extraction Based on Pattern Discovery, *In Proceedings of the World Wide Web Conference*. New York: ACM Press, 681–688.

Chang, C., Kayed, M., Girgis, R. 2006. A Survey of Web Information Extraction Systems, *IEEE Transactions on Knowledge and Data Engineering* 18(10): 1411–1428.

Chang, C., Kuo, S.-C. 2004. OLERA : Semisupervised Web-Data Extraction, *IEEE intelligent Systems* 19(6): 56–64.

Clark, J., Derose, S., Corp, I. 1999. XML Path Language (Xpath). [Online]. [Cited 11 March 2014]. Available from Internet: <Http://Www.W3.Org/TR/Xpath/>.

Connotate. 2012. Web Data Collection & Monitoring Solutions. [Online]. [Cited 11 March 2014]. Available from Internet: <http://www.Connotate.com/Solutions>.

Crescenzi, V. 2001. Roadrunner: towards Automatic Data Extraction from Large Web Sites, *In Proceedings of the International Conference on Very Large Data Bases (VLDB)*. San Francisco: Morgan Kaufmann Publishers inc., 109–118.

Crescenzi, V., Merialdo, P., Missier, P. 2005. Clustering Web Pages Based on Their Structure, *Data & Knowledge Engineering* 54(3): 279–299.

Crescenzi, V., Merialdo, P., Qiu, D., Ingegneria, D., Roma, S. 2013. A Framework for Learning Web Wrappers from the Crowd, *In Proceedings of the World Wide Web Conference*. New York: ACM Press, 261–271.

Dalvi, N., Bohannon, P. 2009. Robust Web Extraction: An Approach Based on A Probabilistic Tree-Edit Model, *In ACM SIGMOD International Conference on Management of Data*. 335–348.

Dalvi, N., Kumar, R., Pang, B., Ramakrishnan, R., tomkins, A., Bohannon, P., Keerthi, S., Merugu, S. 2009. A Web of Concepts, *In Proceedings of the Twenty-Eighth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. New York: ACM Press, 1–12.

Dalvi, N., Kumar, R., Soliman, M. 2011. Automatic Wrappers for Large Scale Web Extraction, *In Proceedings of the VLDB Endowment*. VLDB Endowment, 219–230.

Dalvi, N., Machanavajjhala, A., Pang, B. 2012. An Analysis of Structured Data on the Web, *Proceedings of the VLDB Endowment* 5(7): 680–691.

Damaševičius, R. 2009. Automatic Generation of Concept Taxonomies from Web Search Data Using Support Vector Machine, *In Proc. of the 5th International Conference on Web Information Systems and Technologies WEBIST 2009*. New York: Springer, 673–680.

Dean, J., Henzinger, M. 1999. Finding Related Pages in the World Wide Web, *Computer Networks* 11(31): 1467–1479.

Demaine, E., Mozes, S. 2007. An Optimal Decomposition Algorithm for Tree Edit Distance*, In Automata, Languages and Programming* : 146–157.

Diligenti, M., Coetzee, F.M., Lawrence, S., Giles, C.L., Gori, M. 2000. Focused Crawling Using Context Graphs*, In Proceedings of the VLDB*. San Francisco: Morgan Kaufmann Publishers inc., 527–534.

Doan, A., Halevy, A., Zachary, I. 2013. Principles of Data integration. Amsterdam: Morgan Kaufmann, ISBN 9780124160446.

Elmeleegy, H., Madhavan, J., Halevy, A. 2011. Harvesting Relational Tables from Lists on the Web*, the VLDB Journal* 20(2): 209–226.

Embley, D., Campbell, D., Jiang, Y. 1999. Conceptual-Model-Based Data Extraction from Multiple-Record Web Pages*, Data & Knowledge Engineering* 31(3): 227–251.

Etzioni, O., Fader, A., Christensen, J. 2011. Open Information Extraction: the Second Generation*, In Proceedings of the International Joint Conference on Artificial intelligence (IJCAI)*. San Francisco: AAAI Press, 3–10.

Ferrara, E., Meo, P.D.E., Fiumara, G., Baumgartner, R. 2012. Web Data Extraction, Applications and Techniques : A Survey*, Arxiv* 1207(0246): 1–48.

Fleisher, C.S., Bensoussan, B.E. 2003. Strategic and Competitive Analysis: Methods and Techniques for Analyzing Business Competition. ISBN 9780130888525.

Freitag, D. 2000. Machine Learning for Information Extraction in Informal Domains*, Machine Learning* 39(2-3): 169–202.

Furche, T., Gottlob, G., Grasso, G. 2012a. AMBER: Automatic Supervision for Multi-Attribute Extraction*, Arxiv Preprint* 1210(5984): 1–22.

Furche, T., Gottlob, G., Grasso, G., Gunes, Ö., Guo, X., Kravchenko, A., Orsi, G., Schallhart, C., Sellers, A., Wang, C. 2012b. DIADEM : Domain-Centric , Intelligent , Automated Data Extraction Methodology Categories and Subject Descriptors*, In Proceedings of the World Wide Web Conference*. New York: ACM Press, 267–270.

Furche, T., Gottlob, G., Grasso, G., Schallhart, C., Sellers, A., Foy, C. 2011. Oxpath : A Language for Scalable , Memory-Efficient Data Extraction from Web Applications By Scenario : History Books on Seattle to Extract History Books on Seattle Currently Offered on Amazon *., Proceedings of the VLDB Endowment* 4(7): 1016–1027.

Gonzalez, H., Halevy, A., Jensen, C. 2010. Google Fusion Tables: Web-Centered Data Management and Collaboration*, In Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*. New York: ACM Press, 1061–1066.

Gottron, T. 2008. Clustering Template Based Web Documents*, Advances in Information Retrieval* : 40–51.

Gulhane, P., Madaan, A., Mehta, R., Ramamirtham, J., Rastogi, R., Satpal, S., Sengamedu, S., Tengli, A., Tiwari, C. 2011. Web-Scale Information Extraction with Vertex*, In ICDE*. 1209–1220.

Hammer, J., Mchugh, J., Garcia-Molin, H. 1997. Semistructured Data: the TSIMMIS Experience, *In Proceedings of the First East-European Conference on Advances in Databases and Information Systems*. Swinton: British Computer Society, 1–22.

Henriksson, A., Moen, H., Skeppstedt, M., Eklund, A., Daudaravi˘, V., Hassel, M. 2006. Synonym Extraction of Medical Terms from Clinical Text Using Combinations of Word Space Models, *In Proceedings of the International Symposium on Semantic Mining in Biomedicine*. London: Biomed Central, 10–17.

Hernández, I., Rivero, C.R., Ruiz, D., Corchuelo, R. 2012. A Statistical Approach to URL-Based Web Page Clustering, *In Proceedings of the World Wide Web Conference*. New York: ACM Press, 525–526.

Hong, J.L., Siew, E.-G., Egerton, S. 2010. Information Extraction for Search Engines Using Fast Heuristic Techniques, *Data & Knowledge Engineering* 69(2): 169–196.

Huck, G., Fankhauser, P., Aberer, K., Neuhold, E.J. 1998. Jedi: Extracting and Synthesizing Information from the Web, *In Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems*. Washington: IEEE Computer Society, 32–43.

Yamada, Y., Craswell, N. 2004. Testbed for Information Extraction from Deep Web, *In Proceedings of the World Wide Web Conference*. New York: ACM Press, 346–347.

Yang, W.U.U. 1991. Identifying Syntactic Differences Between Two Programs, *Software - Practise and Experience* 21(JULY): 739–755.

Jindal, N., Bing, L. 2010. A Generalized Tree Matching Algorithm Considering Nested Lists for Web Data Extraction, *In Proceedings of the SIAM International Conference on Data Mining*. Philadelphia: SIAM, 930–941.

Joshi, S., Agrawal, N., Krishnapuram, R., Negi, S. 2003. A Bag of Paths Model for Measuring Structural Similarity in Web Documents, *In Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York: ACM Press, 577–582.

Juntarung, N., Ussahawanitchakit, P. 2008. Knowledge Management Capability, Market intelligence, and Performance: An Empirical investigation of Electronic Businesses in Thailand, *International Journal of Business Research* 8(3): 69–80.

Kayed, M., Chang, C. 2010. Fivatech : Page-Level Web Data Extraction from Template Pages, *IEEE Transactions on Knowledge and Data Engineering* 22(2): 249–263.

Kannan, N. 2010. online Price intelligence for Companies with Real-Time Changes!! [Online]. [Cited 11 March 2014]. Available from Internet: <http://www.ebizq.net/Blogs/Nari/2010/05/Online_Price_Intelligence_For.Php>.

Kaušas, V., Zuokas, D., Medelis, Ž., Krilavičius, T. 2010. Application of Bootstrap Techniques for Police Summaries Retrieval, *In 3rd National Young Scientists Conference of the Lithuanian OR Society*.

Kesteren, A. Van. 2011. CSSOM View Module. [Online]. [Cited 11 March 2014]. Available from Internet: <http://www.W3.Org/TR/Cssom-View/>.

Krilavičius, T., Medelis, Ž., Kapočiūtė-Dzikienė, J., Žalandauskas, T. 2012. News Media Analysis Using Focused Crawl and Natural Language Processing, *In Proceedings of the 19th International Conference on Information and Software Technologies*. New York: Springer, 48–61.

Kushmerick, N. 1997. Wrapper induction for Information Extraction (Doctoral Dissertation). [Online]. [Cited 11 March 2014]. Available from Internet: <http://www.Icst.Pku.Edu.Cn/Course/Mining/11-12spring/参考文献/10-01 Wrapper Induction for Information Extraction.Pdf>.

Laender, A., Ribeiro-Neto, B., Da Silva, A., Silva, E. 2000. Representing Web Data As Complex Objects, *Electronic Commerce and Web Technologies* : 216–228.

Laender, A., Ribeiro-Neto, B., Silva, A. Da. 2002a. A Brief Survey of Web Data Extraction tools, *In Proceedings of the ACM SIGMOD International Conference on Management of Data*. New York: ACM Press, 84–93.

Laender, A., Ribeiro-Neto, B., Silva, A. Da. 2002b. Debye–Data Extraction By Example, *Data & Knowledge Engineering* 40(2): 121–154.

Lam, M.I., Gong, Z. 2005. Web Information Extraction, *In Proceedings of the IEEE International Conference on Information Acquisition*. New York: IEEE Computer Society, 1–6.

Laukaitis, A., Vasilecas, O. 2008. Multi-Alignment Templates Induction, *Informatica* 19(4): 535–554.

Lin, C., Yu, Y., Han, J., Liu, B. 2010. Hierarchical Web-Page Clustering Via in-Page and Cross-Page Link Structures, *Advances in Knowledge Discovery and Data Mining* : 222–229.

Ling, L., Pu, C., Han, W. 2000. XWRAP: An XML-Enabled Wrapper Construction System for Web Information Sources, *In Proceedings of the 16th International Conference on Data Engineering*. Washington: IEEE Computer Society, 611–621.

Liu, B. 2005. NET − A System for Extracting Web Data from Flat and Nested Data Records, *In Proceedings of the International Conference on Web Information System Engineering*. New York: Springer, 487–495.

Liu, B., Grossman, R., Zhai, Y. 2003. Mining Data Records in Web Pages, *In Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York: ACM Press, 601–606.

Liu, W., Meng, X., Meng, W. 2010. Vide : A Vision-Based Approach for Deep Web Data Extraction, *IEEE Transactions on Knowledge and Data Engineering* 22(3): 447–460.

Lönnqvist, A., Pirttimäki, V. 2006. the Measurement of Business intelligence, *Information Systems Management* 23(1): 32–40.

Madhavan, J., Halevy, A. 2009. Harnessing the Deep Web : Present and Future, *Arxiv Preprint* 0909(1785): 1–6.

Madhavan, J., Jeffery, S.R., Cohen, S., Dong, X.L., Ko, D., Yu, C., Halevy, A. 2007. Web-Scale Data integration : You Can only Afford to Pay As You Go*, In Proceedings of the Biennial Conference on innovative Data Systems Research (CIDR)*. New York: SIGMOD, 342–350.

Manku, G.S., Jain, A., Das Sarma, A. 2007. Detecting Near-Duplicates for Web Crawling*, In Proceedings of the World Wide Web Conference*. New York: ACM Press, 141–150.

Miao, G., Tatemura, J., Hsiung, W. 2009. Extracting Data Records from the Web Using Tag Path Clustering*, In Proceedings of the World Wide Web Conference*. New York: ACM Press, 981–990.

Myllymaki, J., Jackson, J. 2002. IBM Research Report Robust Web Data Extraction with XML Path Expressions*, Technical Report, IBM*.

Muslea, I., Minton, S., Knoblock, C.A. 2001. Hierarchical Wrapper Induction for Semistructured Information Sources*, Autonomous Agents and Multi-Agent Systems* 4(1-2): 93–114.

Najork, M., Wiener, J. 2001. Breadth-First Crawling Yields High-Quality Pages*, In Proceedings of the World Wide Web Conference*. New York: ACM Press, 114–118.

Nguyen, H., Fuxman, A., Paparizos, S. 2011. Synthesizing Products for online Catalogs*, Proceedings of the VLDB Endowment* 4(7): 409–418.

Nie, Z., Wen, J. 2007. Object-Level Vertical Search*, In Proceedings of the Biennial Conference on innovative Data Systems Research (CIDR)*. New York: SIGMOD, 235–246.

Nierman, A., Jagadish, H. 2002. Evaluating Structural Similarity in XML Documents*, Webdb* 2: 61–66.

Normantas, K., Vasilecas, O. 2012. Extracting Business Rules from Existing Enterprise Software System*, In Proceedings of the 18th International Conference on Information and Software Technologies*. New York: Springer, 482–496.

Normantas, K., Vasilecas, O. 2013. Normantas, Kęstutis Vasilecas, Olegas*, Baltic Journal of Modern Computing (BJMC)* 1(1-2): 29–51.

Paehl, D. 2012. HTML Tidy Library Project Table of Contents. [Online]. [Cited 11 March 2014]. Available from Internet: <http://tidy.sourceforge.net/>.

Paradauskas, B., Laurikaitis, A. 2006. Business Knowledge Extraction from Legacy Information Systems*, INFORMATION TECHNOLOGY and CONTROL* 35(3): 214–221.

Pisa, U., Informatica, D., Signorini, A. 2005. the indexable Web Is More Than 11.5 Billion Pages*, In in Proceedings of World Wide Web Conference*. 902–903.

Raposo, J., Pan, A., Álvarez, M., Hidalgo, J. 2007. Automatically Maintaining Wrappers for Semi-Structured Web Sources*, Data & Knowledge Engineering* 61(2): 331–358.

Sahuguet, A., Azavant, F. 2001. Building intelligent Web Applications Using Lightweight Wrappers*, Data & Knowledge Engineering* 36(3): 283–316.

Simon, K. 2005. Viper: Augmenting Automatic Information Extraction with Visual Perceptions, *In Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*. New York: ACM Press, 381–388.

Skersys, T., Butleris, R., Kapocius, K., Vileiniskis, T. 2013. An Approach for Extracting Business Vocabularies from Business Process Models, *INFORMATION TECHNOLOGY and CONTROL* 42(2): 178–190.

Sleiman, H. A., Corchuelo, R. 2013. A Survey on Region Extractors from Web Documents, *IEEE Transactions on Knowledge and Data Engineering* 25(9): 1960–1981.

Small, H. 1973. Co-Citation in the Scientific Literature- A New Measure of the Relationship Between Two Documents.Pdf, *Journal of the American Society for Information Science* 4(24): 28–31.

Soderland, S. 1999. Learning Information Extraction Rules for Semi-Structured and Free Text, *Machine Learning Learn.* 34(1-3): 233–272.

Spertus, E. 1997. Parasite: Mining Structural Information on the Web, *Computer Networks and ISDN Systems* 29(8): 587–595.

Su, W., Wang, J. 2009. ODE: ontology-Assisted Data Extraction, *ACM Transactions on Database Systems* 34(2): 1–12.

Su, W., Wang, J., Lochovsky, F.H., Liu, Y. 2011. Combining Tag and Value Similarity for Data Extraction and Alignment, *IEEE Transactions on Knowledge and Data Engineering* 24(7): 1186–1200.

Suchanek, F., Kasneci, G., Weikum, G. 2007. Yago: A Core of Semantic Knowledge, *In in Proceedings of World Wide Web Conference*. 697–706.

Tai, K. 1979. the Tree-To-Tree Correction Problem, *Journal of the ACM (JACM)* 26(3): 422–433.

Thomsen, J.G. 2013. Consistency in the World Wide Web : Specification, Verification, and Evaluation (Doctoral Dissertation). [Online]. [Cited 11 March 2014]. Available from Internet: <http://Pure.Au.Dk/Portal/En/Publications/Consistency-In-The-World-Wide-Web(42e089cf-078f-438d-A027-Aeeef742aad9).html>.

Walther, M. 2012. Unsupervised Extraction of Product Information from Semi-Structured Sources, *In Proceedings of the IEEE 13th International Symposium on Computational intelligence and Informatics*. New York: IEEE Computer Society, 257–262.

Wang, J., Lochovsky, F.H. 2003. Data Extraction and Label Assignment for Web Databases, *In Proceedings of the World Wide Web Conference*. New York: ACM Press, 187–196.

Weikum, G., Theobald, M. 2010. from Information to Knowledge: Harvesting Entities and Relationships from Web Sources, *In Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. New York: ACM Press, 65–76.

Zhai, Y. 2005. Web data extraction based on partial tree alignment, *In Proceedings of the World Wide Web Conference*. New York: ACM Press, 76–85.

Zhai, Y., Liu, B. 2006. Structured Data Extraction from the Web Based on Partial Tree Alignment, *IEEE Transactions on Knowledge and Data Engineering* 18(12): 1614–1628.

Zhang, K., Shasha, D. 1989. Simple Fast Algorithms for the Editing Distance between Trees and Related Problems, *SIAM Journal on Computing* 18(6): 1245–1262.

Zhao, H., Meng, W., Wu, Z., Raghavan, V. 2005. Fully Automatic Wrapper Generation for Search Engines, *In Proceedings of the World Wide Web Conference*. New York: ACM Press, 66–75.

A List of Publications by the Author on the Topic of the Dissertation

Papers in the Reviewed Scientific Journals

Grigalis, T.; Čenys, A. 2014a. Unsupervised Structured Data Extraction from Template-generated Web Pages, *Journal of Universal Computer Science* 20(2): 169–192. ISSN 0948-6968. (THOMSON JCR 2012: 0.762)

Grigalis, T.; Čenys, A. 2014b. Using XPaths of Inbound Links to Cluster Template-Generated Web Pages, *Computer Science and Information Systems* 11(1): 111–131. ISSN 1820-0214. (THOMSON JCR 2012: 0.549)

Grigalis, T.; Čenys, A. 2013. State-of-the-art Web Data Extraction Systems for Online Business Intelligence, *Information sciences* 64: 145–155. ISSN 1392-0561.

Grigalis, T.; Marozas, L.; Radvilavičius, L. 2012a. Analysis of Automated Modern Web Crawling and Testing Tools and Their Possible Employment for Information Extraction, *Science – future of Lithuania* 4(1): 31–34. ISSN 2029-2341.

Other Papers

Grigalis, T. 2013. Towards Web-scale Structured Web Data Extraction, *in Proceedings of the 6th ACM International Conference on Web Search and Data Mining, Rome, Italy.* New York: ACM, 753–758.

Grigalis, T.; Čenys, A. 2012. Generating XPath Expressions for Structured Web Data Record Segmentation, *in Proceedings of the 18th International Conference on Information and software technologies, Kaunas, Lithuania.* Communications in Computer and Information Science, vol. 319. New York: Springer, 38–47. (ISI Proceedings)

Grigalis, T., Radvilavičius, L., Čenys, A., Gordevičius, J. 2012b. Clustering Visually Similar Web Page Elements for Structured Web Data Extraction*, in Proceedings of the 12th International Conference on Web Engineering, Berlin, Germany.* Lecture Notes in Computer Science, vol. 7387. New York: Springer, 435–438.

Grigalis, T. 2012. Towards Automatic Structured Web Data Extraction System, *in Proceedings of the 10th International Baltic Conference on Databases and Information Systems, Vilnius, Lithuania.* Vilnius: Žara, 197–201.

Summary in Lithuanian

Įvadas

Problemos formulavimas

Struktūrizuoti duomenys internete dažniausiai randami tinklalapiuose sugeneruotuose pagal šablonus (Cafarella *et al.* 2011). Paprastai, naršant tokio tipo tinklalapius, kiekvienos užklausos metu yra kreipiamasi į duomenų bazę ir iš jos išrenkami atitinkami struktūrizuoti duomenys. Naudojant iš anksto paruoštus šablonus, šie duomenys automatiškai integruojami į naršomą tinklalapį ir atvaizduojami internetinės svetainės naršytojui.

Internete yra tūkstančiai skirtingos struktūros ir dizaino tinklalapių. Siekis automatiškai atpažinti iš anksto nežinomos struktūros tinklalapius ir išgauti juose esančius struktūrizuotus duomenis yra itin sudėtinga ir daug laiko reikalaujanti problema, kuri tradiciškai yra sprendžiama informacijos gavybos srityje (Bohannon *et al.* 2012; Dalvi *et al.* 2011; Furche *et al.* 2012b). Tradicinės informacijos rinkimo iš tinklalapių priemonės aprašomos duomenų bazių ir duomenų gavybos tyrimų bendruomenėse yra dažniausiai grįstos rankiniu žmogaus darbu, kuomet rankiniu būdu yra rašomos duomenis išrenkančios taisyklės iš konkrečių internetinių šaltinių (Bohannon *et al.* 2012). Ir nors yra siūlymų išgauti šiuos duomenis automatiškai (Crescenzi 2001; Liu *et al.* 2010; Zhai, Liu 2006; Zhao *et al.* 2005), tačiau, deja, dauguma pažangiausių metodų nepasiekia tokio tikslumo, kad būtų galima juos įtraukti į verslo aplinkoje naudojamas programas

(Bohannon *et al.* 2012). Tad struktūrizuotų duomenų gavyba interneto mastu išlieka iki šiol realiai neišspręsta problema (Blanco *et al.* 2011).

Šioje disertacijoje yra moksliškai tyrinėjamas struktūrizuotų duomenų išgavimas iš tinklalapių, sugeneruotų pagal šablonus. Yra siūlomi du nauji metodai, kurie turi potencialo būti panaudoti kuriant automatines duomenų gavybos sistemas interneto mastu.

Darbo aktualumas

Internete gausu informacijos įvairiausia tematika. Gebėjimas kuo didesnę dalį informacijos surasti, išrinkti ir tarpusavyje suintegruoti būtų labai vertingas (Madhavan *et al.* 2007). Išties, tai galėtų stipriai technologiškai pastūmėti pirmyn paieškos internete sistemas, kuomet joms paprastas tekstas iš paprastų žodžių įgytų didesnę semantinę prasmę (Dalvi *et al.* 2009). Dabar dauguma šiuolaikinių paieškos internete sistemų tinklalapius mato kaip tarpusavyje susijusius žodžių kratinius. Vartotojai ieškodami dominančios informacijos internete suveda į paieškos sistemas raktažodžius ir kaip atsakymą gauna eilę nuorodų į tinklalapius, kuriuose tie žodžiai buvo paminėti. Todėl, norėdami gauti atsakymą į mus dominantį klausimą, turime atverti gautas internetines nuorodas ir patys ieškoti atsakymo tinklalapių turinyje, kur ir yra tikroji informacinė vertė (Dalvi *et al.* 2009). Jei paieškos sistemos gebėtų išrinkti tinklalapiuose esančią informaciją ir ją semantiškai apdoroti, tai mes, kaip paieškos sistemų vartotojai, galėtume vietoj nuorodų iškart gauti konkrečią informaciją, kaip kad produktų, skrydžių ar knygų sąrašus su kainomis, filmus ir jų įvertinimus, ir pan. (Weikum, Theobald 2010). Taip pat, interneto masto duomenų išgavimas galėtų pagreitinti duomenų iš daugybės skirtingų šaltinių integraciją. Tai leistų daug sparčiau kurti visaapimančias žinių bazes. Ilgai siekiamas paieškos sistemų tikslas – grąžinti konkrečią informaciją kaip atsakymą, o ne tik nuorodas į tinklalapius – būtų daug lengviau įgyvendinamas (Cafarella *et al.* 2011). Taigi automatinis duomenų išgavimas viso interneto mastu galėtų stipriai patobulinti šiandieninę paiešką ir patį naršymą internete.

Automatinis duomenų išgavimas yra, be abejo, aktualus ir verslo organizacijoms. Šiuolaikinės verslo organizacijos sėkmė priklauso nuo sugebėjimo atitinkamai reaguoti į nuolat besikeičiančia konkurencinę aplinką (Connotate 2012). Daug kompanijų renka informaciją iš tinklalapių ir tokia veikla priskiriama internetinei verslo analitikai (Baumgartner *et al.* 2009b). Pagrindinis internete veikiančios verslo analitikos sistemos tikslas yra rinkti vertingą informaciją iš daugybės skirtingų internetinių šaltinių ir tokiu būdu padėti verslo organizacijai priimti tinkamus sprendimus ir įgyti konkurencinį pranašumą. Tačiau informacijos rinkimas iš daugybės internetinių šaltinių yra ne tik sudėtinga problema, kuomet informaciją renkančios sistemos turi gerai veikti su itin technologiškai sudėtingais tinklalapiais, bet ir daug rankinio darbo reikalaujanti veikla. Kaip žinome, internetinės svetainės yra skirtingo dizaino, t. y. struktūriškai ir vizualiai skirtingos. Jeigu organizacija siekia surinkti informaciją iš daugybės skirtingų šaltinių, ši užduotis tampa sunkiai įgyvendinama dėl itin didelių rankinio programavimo kaštų. Net ir dalinis duomenų rinkimo automatizavimas galėtų padėti organizacijoms taupyti kaštus (Ferrara *et al.* 2012), o pilnai automatizuoti duomenų surinkimo sprendimai padėtų įgyvendinti itin ambicingus siekius rinkti informaciją iš šimtų ar tūkstančių skirtingų tinklalapių. Tačiau automatiniai internetinių duomenų rinkimo sprendimai yra vis dar

aktyviai tyrinėjami ir trūksta realiai veikiančių sistemų. Tokios automatinės sistemos galėtų būti integruotos ir į jau esamas duomenų rinkimo sistemas, ir automatiškai taisyti žmogaus rankiniu būdu rašytas duomenų išrinkimo taisykles jei pasikeičia šaltinio struktūra (Dalvi, Bohannon 2009).

Taigi, kaip matoma, automatinis struktūrizuotų duomenų išgavimas stipriai patobulintų šiuolaikines paieškos internete sistemas, padėtų kompanijoms sumažinti duomenų rinkimo kaštus ir padidinti savo konkurencinį pranašumą.

Tyrimų objektas

Disertacijos tyrimų objektas – struktūrizuotų duomenų išgavimas iš tinklalapių sugeneruotų pagal šablonus.

Darbo tikslas

Mokslinio darbo tikslas – pasiūlyti naują efektyvesnį metodą, skirtą išgauti struktūrizuotus duomenis iš tinklalapių, sugeneruotų pagal šablonus.

Darbo uždaviniai

Darbo tikslui pasiekti ir mokslinei problemai spręsti darbe buvo iškelti šie uždaviniai:

1. Išanalizuoti šiuolaikinius duomenų išgavimo metodus.
2. Išanalizuoti technologiškai pažangius šiuolaikinius tinklalapius, kuriuose yra randama struktūrizuotų duomenų.
3. Pasiūlyti metodą, skirtą išgauti struktūrizuotus duomenis iš šiuolaikinių tinklalapių, sugeneruotų pagal šablonus.
4. Pasiūlyti metodą, skirtą klasterizuoti struktūriškai panašius tinklalapius, sugeneruotus pagal šablonus.
5. Eksperimentiškai ištirti pasiūlytus metodus ir juos kiekybiškai palyginti su kitų autorių metodais.

Tyrimų metodika

Darbe taikyti šie tyrimų metodai:

1. Pažintinio tyrimo metodas, naudotas siekiant įsigilinti ir išanalizuoti mokslinio tyrimo objektą ir siekiant atlikti literatūros analizę.
2. Konstruktyvinis tyrimo metodas, naudotas konstruojant ir eksperimentiškai išbandant šioje disertacijoje siūlomus naujus duomenų išgavimo ir tinklalapių klasterizavimo metodus. Pasiūlyti du nauji metodai buvo realizuoti kaip prototipai naudojant Perl, Python ir JavaScript programavimo kalbas.

Darbo mokslinis naujumas

Darbo mokslinis naujumas pagrįstas šiais rezultatais:

1. Pasiūlytas naujas metodas pavadintas ClustVX, kuris yra skirtas struktūrizuotų duomenų išgavimui iš tinklalapių, sugeneruotų pagal šablonus. Metodas yra grįstas struktūriškai ir vizualiai panašių tinklalapio elementų klasterizacija. Šis metodas leidžia išgauti duomenis ir iš technologiškai sudėtingų šiuolaikinių tinklalapių, nes prieš išgaunant duomenis tinklalapis yra atvaizduojamas naudojant šiuolaikinę interneto naršyklę. Eksperimentiniais tyrimais parodyta, jog ClustVX metodas pasiekia didesnį nei 98% tikslumą ir atkuriamumą ir tuo yra efektyvesnis negu kitų autorių metodai.
2. Pasiūlytas ClustVX metodas yra nejautrus jokiai teminiai sričiai, ir jam nereikia turėti išankstinės informacijos apie išgaunamus struktūrizuotus duomenis. Metodas veikia pilnai automatiškai ir nereikalauja jokio rankinio žmogaus darbo, todėl gali būti panaudotas duomenų išgavimo sistemose interneto mastu.
3. Pasiūlytas naujas metodas pavadintas UXClust, skirtas sparčiai klasterizuoti panašios struktūros tinklalapius, kurie yra sugeneruoti pagal šablonus. Šis metodas išnaudoja nuorodų į vidinius tinklalapius XPath adresus tam, kad reikšmingai pagreitintų panašios struktūros tinklalapių klasterizaciją. Eksperimentiškais tyrimais parodyta, kad, naudojant šį metodą, galima suklasterizuoti daugiau kaip vieną milijoną tinklalapių per nepilnas 4 minutes ir kartu išlaikyti didesnį negu 90% tikslumą ir atkuriamumą. Šie rezultatai gerokai pranoksta kitų autorių metodų efektyvumą.

Darbo rezultatų praktinė reikšmė

Pasiūlytas naujas metodas, skirtas struktūrizuotų duomenų išgavimui, yra tinkamas naudoti duomenims automatiškai išgauti iš tinklalapių sugeneruotų pagal šablonus. Kadangi šiam metodui nereikalingas rankinis žmogaus darbas ar išankstinis apmokymas, jis gali padėti verslo organizacijoms žymiai sumažinti duomenų rinkimo iš internetinių tinklalapių kaštus. Taip pat šis metodas yra tinkamas automatiškai generuoti XPath kalba grįstas duomenų išgavimo taisykles, kurios gali būti panaudotos kitose duomenų rinkimo sistemose net ir interneto mastu.

Antrasis pasiūlytas metodas skirtas tinklalapių klasterizavimui yra tinkamas naudoti klasterizuojant struktūriškai panašius tinklalapius, sugeneruotus pagal šablonus. Kadangi šis metodas yra itin spartus, jis yra tinkamas naudoti duomenų rinkimo sistemose interneto mastu, kuriose yra aktualus tinklalapių klasterizavimo uždavinys.

Ginamieji teiginiai

1. Klasteriai su tarpusavyje vizualiai ir struktūriškai panašių tinklalapio elementų XPath adresais gali būti panaudoti nustatant pasikartojančią struktūrą, kurioje yra užkoduoti struktūrizuoti duomenys, randami pagal šablonus sugeneruotuose tinklalapiuose.

2. Nuorodų tarp vidinių internetinės svetainės tinklalapių XPath adresai gali būti panaudoti reikšmingai pagreitinti struktūriškai panašių pagal šablonus sugeneruotų tinklalapių klasterizaciją.

Darbo rezultatų aprobavimas

Disertacijos tema paskelbti 8 moksliniai straipsniai. Keturi iš jų yra publikuoti recenzuojamuose mokslo žurnaluose, iš kurių du yra įtraukti į Thompson Reuters ISI Web of Science duomenų bazę ir turi citavimo indeksą.

Disertacijos rezultatai buvo pristatyti šešiose tarptautinėse mokslinėse konferencijose ir vienoje tarptautinėje mokslinėje vasaros mokykloje:

- 10th International Baltic Conference on Databases and Information Systems (Baltic DB&IS 2012). Liepos 8–11, 2012, Vilnius, Lietuva;
- 12th International Conference on Web Engineering (ICWE 2012). Liepos 23–27, 2012, Berlynas, Vokietija;
- 18th International Conference on Information and Software Technologies (ICIST 2012). Rugsėjo 13–14, 2012, Kaunas, Lietuva;
- 6th ACM International Conference on Web Search and Data Mining (WSDM 2013). Vasario 4–8, Roma, Italija;
- 3rd Workshop on Data Extraction and Object Search (DEOS 2013). Liepos 6–8, 2013, Oksordas, Didžioji Britanija;
- 12th Estonian Summer School on Computer and Systems Science (ESSCaSS 2013). Rugpjūčio 18–22, 2013, Voore, Estija;
- 4th International Workshop on Data Analysis Methods for Software Systems. Gruodžio 5–7, 2013, Druskininkai, Lietuva.

Disertacijos struktūra

Disertaciją sudaro įvadas, keturi pagrindiniai skyriai, bendrosios išvados, literatūros šaltinių sąrašas, autoriaus publikacijų disertacijos tema sąrašas, santrauka lietuvių kalba. Darbo apimtis – 123 puslapiai neskaitant priedų, tekste yra 22 formulės, 37 paveikslai ir 20 lentelių. Rašant disertaciją buvo panaudoti 116 literatūros šaltinių.

1. Struktūrizuotų duomenų išgavimas, metodai ir jų panaudojimas

Šiame skyriuje yra pristatoma struktūrizuotų internetinių duomenų išgavimo mokslinė problema, analizuojama literatūra, kitų autorių siūlomi metodai, apžvelgiami praktiniai struktūrizuotų duomenų išgavimo metodų pritaikymo aspektai.

Struktūrizuotų duomenų išgavimo metodais sprendžiama problema, kaip išrinkti duomenis iš internetinių tinklalapių. Duomenų išgavimo problema gali būti dalinama į dvi dideles sritis: informacijos išgavimas iš natūralios kalbos teksto ir struktūrizuotų duomenų

išgavimas iš tinklalapių (Bing 2012). Esminis skirtumas tarp šių dviejų šakų yra tai, kad didžioji dalis duomenys tinklalapiuose yra užkoduoti naudojant hiperteksto žymėjimo kalbą (HTML) ir kiekviename tinklalapyje išsiskiria savo reguliariu struktūriniu ir vaizdiniu atvaizdavimu (Cafarella *et al.* 2011). Tai ypač matoma, kuomet duomenys atkeliauja iš duomenų bazių ir į tinklalapius yra įrašomi naudojant iš anksto paruoštus šablonus. Tad struktūrizuotų duomenų išgavimas iš tinklalapių sugeneruotų pagal šablonus tuo ir skiriasi nuo informacijos išgavimo iš paprasto teksto, kuris nėra niekaip kitaip struktūrizuotas, negu natūralios kalbos dalimis, sakiniais. Informacijos išgavimo iš paprasto teksto metodai dažniausiai mėgina atpažinti kalbos dalis tekste, ieško ir išrenka realaus pasaulio esybes, ryšius, kitą paprastomis sakinio formomis aprašomą faktinę informaciją (Furche *et al.* 2012a), o struktūrizuotų duomenų išgavimo metodai siekia išrinkti duomenis ir atkurti jų struktūrą taip, kaip tie duomenys buvo saugomi duomenų bazės lentelėse. Informacijos rinkimas iš paprasto teksto dažniausiai susijęs su natūralios kalbos tyrinėjimu (Bing 2012), o šioje disertacijoje yra gilinamasi būtent į struktūrizuotų duomenų išgavimą iš tinklalapių sugeneruotų pagal šablonus.

Struktūrinis ir vaizdinis struktūrizuotų duomenų atvaizdavimo tinklalapiuose reguliarumas yra itin svarbi savybė, nes identifikavus šį reguliarumą galima nuspėti, kokia struktūra HTML kalba užkoduoti duomenys buvo saugomi duomenų bazės lentelėse. Esminiai struktūrizuotų duomenų išgavimo metodų uždaviniai yra šie:

1. Surasti vaizdinį ir struktūrinį duomenų atvaizdavimo tinklalapyje pasikartojamumą.
2. Identifikuoti struktūrizuotus duomenis, atskiriant juos nuo hipertekstinės žymėjimo kalbos.
3. Sugeneruoti duomenis išgaunančias taisykles, skirtas analizuojamam tinklalapiui.
4. Naudojant sugeneruotas taisykles išgauti duomenis iš to paties šablono tinklalapių.

Tradiciškai duomenų surinkimas iš tinklalapių yra sprendžiamas rankiniu būdu rašant duomenų išrinkimo taisykles. Savaime suprantama, jog ši programuotojo darbo reikalaujanti veikla yra daug kainuojanti, ir organizacijos patiria didelius kaštus. Vienas iš būdų klasifikuoti duomenų išgavimo metodus yra pagal tai, kiek rankinio programuotojo darbo jiems yra reikalinga. Todėl dauguma struktūrizuotų duomenų išgavimo metodų gali būti skirstomi į šias tris kategorijas (Bing 2012):

1. Visiškai rankiniai metodai.
2. Pusiau automatiniai metodai.
3. Pilnai automatiniai metodai.

Naudojant vien tik rankinio darbo reikalaujančius metodus (pirma kategorija), žmogus programuotojas pats analizuoja tinklalapio struktūrą, hipertekstinio žymėjimo kodą, ieško pasikartojančių elementų ir suprogramuoja duomenų išrinkimo taisykles (Arocena, Mendelzon 1998; Califf, Mooney 1999; Hammer *et al.* 1997; Laender *et al.* 2000). Palengvinant šį darbą yra sukurta eilė duomenų išrinkimo taisyklių rašymo kalbų bei darbą palengvinančių vartotojo sąsajų. Tačiau rankiniai duomenų išgavimo metodai nėra tinkami rinkti duomenis iš daugybės skirtingų tinklalapių, nes tai reikalauja tiesiog per didelių programavimo kaštų (Bing 2012).

Pusiau automatiniai metodai padeda programuotojui kurti duomenų išgavimo taisykles. Dažniausiai programuotojui yra pateikiama tam tikra programinė aplinka, kurioje atvaizduojamas tinklalapis ir galima rankiniu būdu sužymėti norimus išrinkti duomenis (Baumgartner, Flesca 2001; Britain *et al.* 1998; Chang, Kuo 2004; Chang 2001; Kushmerick 1997; Laender *et al.* 2002). Tuomet programa sugeneruoja taisykles, kurias galima naudoti išgaunant duomenis. Kaip ir galima nuspėti, du pagrindiniai šios kategorijos metodų trūkumai: brangus žmogaus darbo reikalaujantis procesas ir nuolatinis duomenų išgavimo taisyklių taisymas, kuomet pasikeičia tinklalapio dizainas (Gulhane *et al.* 2011; Kushmerick 1997; Raposo *et al.* 2007).

Trečioji duomenų išgavimo kategorija yra pilnai automatiniai metodai, kurių esminis privalumas yra tai, kad jie dažniausiai beveik visiškai nereikalauja žmogaus darbo (Álvarez *et al.* 2008; Crescenzi 2001; Hong *et al.* 2010; Jindal, Bing 2010; Kayed, Chang 2010; Liu 2005; Liu *et al.* 2003, 2010; Simon 2005; Su *et al.* 2011; Zhai, Liu 2006; Zhao *et al.* 2005).

Taigi, rankinio programavimo reikalaujantys metodai negali būti išnaudojami praktikoje rinkti duomenis iš šimtų ar tūkstančių skirtingo dizaino tinklalapių, nes duomenų išgavimo taisyklių kūrimas yra pernelyg brangus organizacijoms. Todėl didžioji dalis tyrimų krypsta į trečiosios kategorijos, t. y. visiškai automatinius duomenų išgavimo metodus (Bohannon *et al.* 2012; Dalvi *et al.* 2011; Elmeleegy *et al.* 2011; Furche *et al.* 2012b; Gulhane *et al.* 2011), kurie gali būti pritaikyti rinkti duomenis iš daugybės tinklalapių interneto mastu. Ne išimtis ir ši disertacija, kurioje tyrinėjama automatinių duomenų išgavimo metodų sritis.

2. Metodas, skirtas išgauti struktūrizuotus duomenis iš tinklalapių sugeneruotų pagal šablonus

Šiame skyriuje pristatomas naujas metodas, skirtas išgauti struktūrizuotus duomenis iš tinklalapių sugeneruotų pagal šablonus. Siūlomas metodas yra pavadintas ClustVX (kilęs iš angliškų žodžių *Clustering Visually similar XPaths*).

Siūlomas metodas remiasi dvejomis pagrindinėmis prielaidomis. Pirma, jog didelė dalis duomenų randamų internete yra pateikiami tinklalapiuose, kurie yra sugeneruoti pagal šablonus, ir į kuriuos duomenys patenka iš duomenų bazių (Cafarella *et al.* 2011). Paveiksle S1(a) yra pateikta būtent tokio tipo tinklalapio iškarpa. Joje matomi trys struktūrizuoti duomenų įrašai (produktus – skaitmeninius fotoaparatus), kurie į tinklalapį yra įkrauti iš duomenų bazės. Visi trys įrašai yra struktūriškai panašūs ir išrikiuoti vienas šalia kito. Tinklalapio vieta, kur randama struktūriškai panašių duomenų įrašų eilė vadinama duomenų zona (Zhai, Liu 2006). Kaip žinoma, tinklalapiai dažniausiai yra parašyti naudojant hipertekstinę žymėjimo kalbą. Šį hipertekstinį programinį kodą dažnai yra paranku vaizduoti medžio struktūros (HTML tree). Tai leidžia lengvai identifikuoti medžio šakas ir jų adresus aprašyti XPath kalba. Taigi, jeigu struktūrizuoti duomenų įrašai yra randami po viena hipertekstinio dokumento medžio šaka, tai jų XPath adresai bus labai panašūs, ir dažnai skirsis tik pošakių indeksai.

Antra, internetinių svetainių dizaineriai kuria tinklalapių šablonus, pritaikytus skaityti žmonėms. Todėl nors kiekvieno tinklalapio dizainas ir struktūrizuotų duomenų išdėstymas ir spalvinis atvaizdavimas skiriasi, tačiau žmogus gana lengvai įžvelgia tvarkingai pasikartojančius duomenis ir lengvai juos perskaito (Miao *et al.* 2009). Duomenys, turintys vienodą semantinę prasmę, tame pačiame tinklalapyje atvaizduojami vienodai (Liu *et al.* 2010). Kaip pavyzdys pateikta ta pati tinklalapio iškarpa S1(a) paveiksle. Matyti, jog kainos ties kiekvienu įrašu (produktu) yra toje pačioje vietoje, paryškintu šriftu, produktų pavadinimai – žemiau paveiksliukų ir t.t.

a) tinklalapio fragmentas su trimis struktūrizuotais duomenų įrašais

Xstring:	htmlbodydivdivdivfonta-Verdana,brown-red;400
$84.95	/html/body/div[3] /div[1] /div/font/a
$174.95	/html/body/div[3] /div[2] /div/font/a
$84.95	/html/body/div[3] /div[3] /div/font/a

b) klasteris su struktūriškai ir vizualiai panašiais tinklalapio elementais

Image 1	Samsung ES80	$84.95	Online Price
Image 2	Fujifilm FinePix T300	$174.95	Online Price
Image 3	Vivitar ViviCam F529	$84.95	Online Price

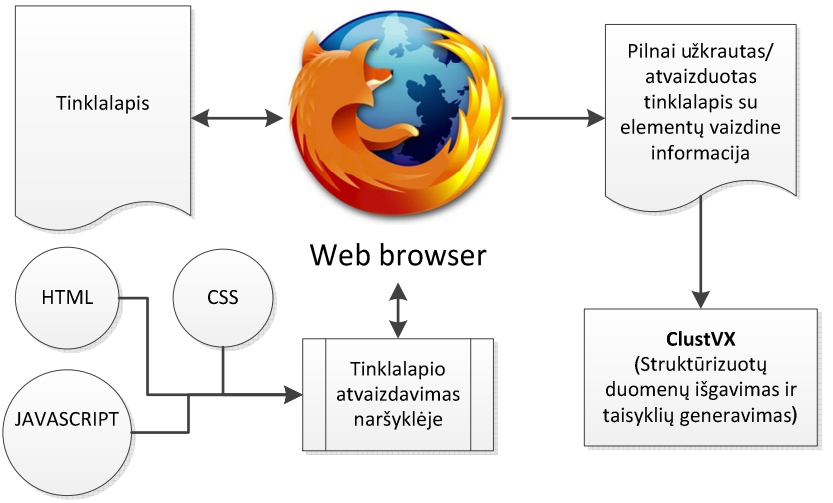
c) išgauti struktūrizuoti duomenys

S1 pav. Struktūrizuotų duomenų išgavimas naudojant ClustVX metodą

Įvardintos dvi esminės prielaidos apie struktūrizuotų duomenų atvaizdavimą tinklalapiuose, t. y. jų struktūrinis ir vizualinis panašumas, įgalino sukurti siūlomą ClustVX metodą, kuris remiasi struktūriškai ir vizualiai panašių elementų klasterizacija. Šis metodas kiekvieną analizuojamame puslapyje matomą elementą pažymi kaip tekstinę eilutę, sudarytą iš XPath adreso (be šakų indeksų) ir atvaizdavimo stiliaus savybių, tokių kaip šrifto dydis, spalva, tipas ir t.t. Ši tekstinė eilutė yra vadinama *Xstring*. Suklasterizavęs tokias eilutes, ClustVX metodas gauna eilę klasterių, iš kurių kiekviename

yra struktūriškai ir vizualiai panašūs elementai. S1(b) paveiksle pavaizduotas vienas toks klasteris, kurį sudaro produktų kainų elementai ir jų XPath adresai. To paties paveikslėlio viršuje, į dešinę nuo "Xstring:", yra pavaizduota ir *Xstring* eilutė, kuri buvo naudojama kaip elementų tarpusavio atstumą nusakantis atributas klasterizavimo metu. Kiekvieną klasterį sudaro tik tokią pačią *Xstring* eilutę turintys elementai. Toliau apdorojant suklasterizuotų elementų XPath adresus yra išvedami duomenų ištraukimo taisyklių rinkiniai, angliškai vadinami „apgaubėjais" (angl. *wrappers*).

S1(c) paveikslo dalyje pavaizduoti jau struktūrizuoti duomenys, kurie gauti įvykdžius automatiškai sugeneruotą „apgaubėją". Šis struktūrizuotas duomenų pateikimas lentele yra artimas duomenų bazėje saugomų duomenų struktūrai.



S2 pav. Prototipinės sistemos su realizuotu ClustVX metodu architektūra

S2 paveiksle pavaizduota realizuoto ClusVX metodo prototipo architektūra. Kaip matyti iš paveikslo, centrinis metodo elementas yra šiuolaikinė interneto naršyklė Mozilla Firefox. Šioje interneto naršyklėje yra užkraunami visi tinklalapiai, iš kurių siekiama išgauti struktūrizuotus duomenis. Tai daroma dėl dviejų priežasčių: pirma, šiuolaikiniai tinklalapiai yra itin sudėtingi ir dažnai duomenys į juos patenka naudojant AJAX (*Asinchroninis JavaScript ir XML programavimas*) technologiją, kuria leidžia jau užkrovus tinklalapį papildomai kreiptis į serverį ir atsisiųsti duomenis. Todėl dažnai sutinkama situacija, kai pagal tinklalapio adresą tiesiai iš serverio atsisiųstas puslapis yra tuščias šablonas, nes nebuvo įvykdytos papildomos asinchroninės užklausos.

Antra, iš pirmo žvilgsnio gana paprasta užduotis – gauti tinklalapio elementų atvaizdavimo stilių (spalvą, dydį ir t.t.) – nėra lengvai įgyvendinama praktikoje, nes stilių aprašantis kodas gali būti įrašomas daugybėje skirtingų vietų, tokių kaip hipertekstinės žymėjimo kalbos elementų atributuose, nutolusiuose CSS (angl. Cascading Style Sheets) failuose, JavaScript kode ir t. t.

Sprendžiant minėtas problemas, t. y. JavaScript kodo įvykdymą ir pilną atvaizdavimą, ir yra naudojama šiuolaikinė interneto naršyklė. Pilnai užkrauto tinklalapio kodas (HTML dokumentas) kartu su elementų atvaizdavimo savybėmis (spalva, šrifto dydžiu, tipu ir t.t.) yra persiunčiamas į tolimesnius ClustVX metode aprašytus žingsnius. Tokiu būdu ClustVX sugeba išgauti duomenis ir iš technologiškai sudėtingų šiuolaikiškų tinklalapių, ir tai dažnai leidžia padidinti metodo efektyvumą lyginant su kitų autorių metodais, kuriuose nenaudojama šiuolaikinė naršyklė.

Sėkminga pasiūlyto ClustVX realizacija pagrindė jo praktinę vertę. Tapo aišku, jog realizuotą prototipą galima patalpinti nutolusiame serveryje ir padaryti jį pasiekiamą per internetą trečiųjų šalių sistemoms. Tokiu būdu metodas gali būti gana lengvai panaudotas kaip sudėtinė kitų duomenis renkančių sistemų dalis.

3. Metodas, skirtas klasterizuoti struktūriškai panašius tinklalapius, sugeneruotus pagal šablonus

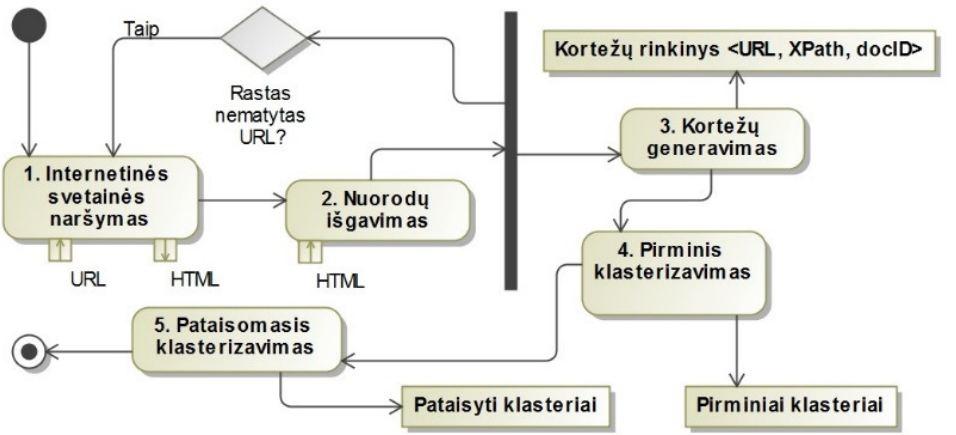
Struktūrizuotų duomenų išgavimo sistemos, apdorojančios itin didelius kiekius tinklalapių, susiduria su problema, kaip atpažinti pagal tą patį šabloną sugeneruotus tinklalapius. Kaip žinia, kiekvienam šablonui galima sugeneruoti duomenis išgaunančias taisykles ir jas naudoti kiekvienam to paties šablono tinklalapiui. Todėl ieškoma sprendimo, kaip sparčiai suklasterizuoti šimtus tūkstančių ar milijonus tinklalapių pagal jų struktūrinį panašumą (Blanco *et al.* 2011).

Nors tinklalapių klasterizavimo pagal jų struktūrinį panašumą problema yra jau gana ilgai tyrinėjama (Blanco *et al.* 2011; Chakrabarti, Mehta 2010; Crescenzi *et al.* 2005; Joshi *et al.* 2003), tačiau dauguma metodų remiasi išskirtinai tinklalapių turinio analize ir naudojami algoritmai yra kvadratinio sudėtingumo. Tai kelia rimtų problemų, nes tokio tipo sprendimai nėra tinkami išgaunant duomenis ir klasterizuojant tinklalapius interneto mastu (Blanco *et al.* 2011). Vienoje duomenų baze paremtoje interneto svetainėje, kurioje tinklalapiai generuojami pagal šablonus, gali būti milijonai pasiekiamų tinklalapių su skirtingais duomenimis. Tarkime, projektas XProj (Aggarwal *et al.* 2007) yra vienas naujausių ir žinomiausių sprendimų, skirtų XML dokumentų klasterizavimui (pritaikoma ir tinklalapiams prarašytiems švariu HTML kodu). Ši sistema naudoja linijinio sudėtingumo algoritmus ir sugeba suklasterizuoti milijoną tinklalapių tik per 20 valandų (Blanco *et al.* 2011).

Šiame skyriuje aprašomas antrasis šioje disertacijoje siūlomas metodas, kuris yra skirtas sparčiai klasterizuoti struktūriškai panašius tinklalapius, sugeneruotus naudojant šablonus. Metodas pavadintas UXClust (kilęs iš angl. žodžių *URL XPath Clustering*), ir jis išnaudoja vidinių interneto svetainės nuorodų XPath adresus tinklalapiuose tam, kad reikšmingai pagreitintų tinklalapių klasterizaciją.

Kaip jau žinome, interneto svetainių dizaineriai pritaiko tinklalapių išvaizdą vartotojams perskaityti. Toje pačioje interneto svetainėje tinklalapiai, sugeneruoti pagal tą patį šabloną, yra labai panašūs ir skiriasi tik informacija bei keliais paveiksliukais (pvz.: produkto pavadinimas, kaina, nuotrauka, aprašymas). Vartotojai lengvai įsimena šabloną ir greitai perskaito bei supranta kiekvieną pagal jį sugeneruotą tinklalapį. Skirtingos

išvaizdos šablonų vienoje interneto svetainėje nėra daug, be to, kiekviename iš šių šablonų yra numatytos konkrečios vietos, kuriose yra talpinamos nuorodos į kitus vidinius svetainės tinklalapius. Taigi, siūlomas UXClust metodas remiasi prielaida, jog toje pačioje šablono vietoje esanti nuoroda kiekviename pagal tą šabloną sugeneruotame tinklalapyje veda į struktūriškai panašius tinklalapius. Šis šablono išvaizdos ir funkcionalumo reguliarumas yra lengvai suprantamas vartotojams, tačiau kartu yra labai gera priemonė nuspėti pagal nuorodą randamo tinklalapio šabloną.



S3 pav. Prototipinės sistemos su realizuotu UXClust metodu veiklos diagrama

Klasterizuodamas XPath adresus, kuriuose randamos nuorodos, UXClust metodas išvengia daug resursų reikalaujančios tinklalapio turinio analizės, dėl kurios kitų autorių metodai yra gerokai lėčiau veikiantys. Žinoma, skirtingi XPath adresai su nuorodomis gali rodyti į tos paties šablono tinklalapius. Tarkime, elektroninės komercijos svetainėje matant produktų sąrašą ir spaudžiant ant bet kurio produkto pavadinimo, nuotraukos ar kainos, greičiausiai bus patenkama į to paties šablono tinklalapį, t. y. detalaus produkto aprašymo tinklalapį. Todėl vienas iš UXClust metodo žingsnių ir yra patikrinti ar nereikia tam tikrų klasterių sujungti.

```
/html/body/div/div/a
/html/body/div/center/table/tbody/tr/td/div/a
/html/body/div/p/div/ul/li
/html/body/div/div/p/div/ul/li
/html/body/footer/div/p
```

S4 pav. Tinklalapį sudarančių elementų XPath rinkino pavyzdys

Pagal siūlomą UXClust metodą realizuoto prototipo veiklos diagrama, matoma S3 paveiksle, ir sudaryta iš šių penkių žingsnių: interneto svetainės naršymo; nuorodų

išgavimo; iš URL, dokumento ID ir XPath adresų, sudarytų kortežų generavimu; pirminio klasterizavimo; pataisomojo klasterizavimo. Penktajame žingsnyje - pataisomojo klasterizavimo metu, ieškoma klasterių, kuriuose esančių tinklalapių struktūrinis panašumas yra didesnis, negu iš anksto nustatyta konstanta. Struktūrinis panašumas tarp dviejų tinklalapių iš skirtingų klasterių yra nustatomas lyginant tų tinklalapių programinio kodo, t. y. HTML, medžio struktūros baigtinių šakų (neturinčių vaikų) XPath adresus (Joshi *et al.* 2003). S4 paveiksle pateikiami tokių XPath adresų pavyzdžiai.

Taigi, tokiu būdu kiekvienas tinklalapis, vadinamas P_i, gali būti laikomas medžio šakų XPath adresų rinkiniu, vadinamu $xp(P_i)$. Tinklalapių panašumas (CPD) yra išreikštas kaip vienodų medžio šakų XPath adresų santykis su didesniojo tinklalapio medžių šakų skaičiumi (Gottron 2008):

$$CPD(P_1, P_2) = 1 - \frac{|xp(P_1) \cap xp(P_2)|}{\max(|xp(P_1)|, |xp(P_2)|)}. \quad (1)$$

Tad išnaudojant vidinių interneto svetainės nuorodų XPath adresus, galima apytiksliai suklasterizuoti tinklalapius pagal jų struktūrinį panašumą. Tokiam klasterizavimui nereikia analizuoti ir lyginti tinklalapių turinio, todėl ženkliai pagreitėja klasterizavimo procesas. Pataisomojo klasterizavimo metu klasteriai, turintys vienodo šablono tinklalapius, yra sujungiami. Tačiau ir pastarojo žingsnio metu yra lyginamas tik kelių tinklalapių turinys (XPath adresai), kas nesumažina UXClust metodo efektyvumo laiko prasme.

4. Pasiūlytų metodų eksperimentiniai tyrimai

Šiame skyriuje aprašomi eksperimentiniai tyrimai, kuriais išbandomi du pasiūlyti metodai, t. y. metodas vadinamas UXClust, kuris skirtas klasterizuoti tinklalapius pagal jų struktūrinį panašumą, ir metodas ClustVX, kuris yra skirtas struktūrizuotiems duomenims išgauti iš tinklalapių, sugeneruotų pagal šablonus. Abu siūlomi metodai buvo realizuoti kaip prototipai. ClustVX realizuotas naudojant PERL ir JavaScript programavimo kalbas, o UXClust – naudojant Python programavimo kalbą. Eksperimentai buvo vykdomi kompiuteryje su Ubuntu 12.04 operacine sistema, Intel® Core™ i7-2670QM CPU @ 2.20 GHz centriniu procesoriumi, 8 GB operatyviąja atmintimi, 7200 RPM standžiuoju disku.

Abu metodai buvo išbandyti su realiais tinklalapiais, lyginant rezultatus su kitų autorių atitinkamų metodų rezultatais. Disertacijoje siūlomas UXClust metodas buvo lygintas su *common XPaths* (Joshi *et al.* 2003) and *pq-grams* (Augsten *et al.* 2005) metodais, o ClustVX su – *G-STM* (Jindal, Bing 2010), *DEPTA* (Zhai, Liu 2006), *FiVaTech* (Kayed, Chang 2010), *MDR* (Liu *et al.* 2003), *ViNTs* (Zhao *et al.* 2005), *CTVS* (Su *et al.* 2011), *DeLa* (Wang, Lochovsky 2003) ir metodu aprašytu (Álvarez *et al.* 2008).

UXClust metodo bandymų atveju buvo sudarytas testinis tinklalapių rinkinys, susidedantis iš daugiau kaip vieno milijono tinklalapių, atsiųstų iš keturiolikos skirtingų internetinių svetainių. Visi šie tinklalapiai sugeneruoti pagal šablonus. S1 lentelėje matyti,

jog bandymams buvo naudota daugiau kaip vienas milijonas tinklalapių, kurie bendrai užėmė 119,1 GB duomenų.

S1 lentelė. Apibendrinti duomenys apie eksperimente naudotų svetainių tinklalapius

Nr.	Svetainė	Tinklalapių	Duomenų kiekis GB	Vidutinis dydis KB
1.	argos.co.uk	111142	7,33	69,20
2.	azon.lt	102313	18,28	187,32
3.	bigbox.lt	120557	27,03	235,12
4.	citylights.com	13698	0,34	26,29
5.	currys.co.uk	9993	0,67	70,42
6.	elshop.lt	90001	2,27	26,44
7.	ikea.com	122686	10,16	86,83
8.	ilterzogirone.it	66404	3,64	57,55
9.	imk.lt	74295	14,46	204,14
10.	iristorante.it	116410	5,35	48,21
11.	kompiutera.lt	21623	1,08	52,24
12.	smartbuy.lt	15638	0,41	27,62
13.	tesco.com	88773	15,64	184,76
14.	varle.lt	117514	12,44	110,96
Vidurkis:		76503	8.51	99,08
Viso:		1071047	119,1	–

Iš kiekvienos internetinės svetainės tinklalapių šablonų buvo pasirinktas vienas konkretus, kuris buvo naudojamas skaičiuojant klasterizavimo efektyvumą. Efektyvumas buvo apibrėžtas trimis rodikliais: tikslumu (angl. *precision*), atkuriamumu (angl. *recall*) ir vykdymo laiku. Skaičiuojant efektyvumą teisingai teigiamu (TT) sprendimu buvo laikomas rezultatas, kuomet metodas priskirdavo du struktūriškai panašius tinklalapius į tą patį klasterį. Teisingai neigiamas (TN) sprendimas reiškė, jog du struktūriškai nepanašūs tinklalapiai priskirti į skirtingus klasterius. Du klaidingi sprendimai buvo įvardinti kaip klaidingai teisingas (KT), kuomet du struktūriškai nepanašūs tinklalapiai priskiriami vienam klasteriui, o klaidingai neteisingas (KN) – kuomet du struktūriškai panašūs tinklalapiai priskiriami skirtingiems klasteriams. Tokiu būdu tikslumas ir atkuriamumas buvo skaičiuojamas pagal šias formules:

$$Tikslumas = \frac{TT}{TT+KT}\ , \tag{2}$$

$$Atkuriamumas = \frac{TT}{TT+KN}. \quad (3)$$

S2 lentelėje matyti eksperimentinių tyrimų rezultatus, kuriuose siūlomo metodo klasterizavimo rezultatai lyginami su dvejais kitų autorių metodų, t. y. pg-Grams ir CP rezultatais. Skliausteliuose esantis skaičius prie metodo pavadinimo nurodo skaičių tinklalapių, kurie buvo panaudoti vertinant atitinkamo metodo efektyvumą. Šis skaičius skiriasi, nes metodų vykdymo laikas ir gebėjimas apdoroti didelį kiekį tinklalapių labai skyrėsi. Siūlomas metodas UXClust dirbo su daugiau kaip milijonu tinklalapių, kai pg-Grams - su 100, o CP - su 1000. Antrojoje lentelės eilutėje „Viso“ reiškia pagal kiek tinklalapių buvo matuojamas tikslumas ir atkuriamumas, T – tai tikslumas, o A – atkuriamumas. Kaip matoma iš rezultatų, siūlomas UXClust metodas nenusileidžia kitiems dviem metodams savo tikslumu ir efektyvumu, o daugeliu atvejų net ir lenkia kitus metodus.

S2 lentelė. Tinklalapių klasterizavimo rezultatai naudojant siūlomą ir du kitų autorių metodus

Nr.	**Svetainė**	**UXClust**			**pg-Grams (100)**			**CP (1000)**		
		Viso	*T*	*A*	*Viso*	*T*	*A*	*Viso*	*T*	*A*
1.	argos.co.uk	30460	0,92	0,72	40	1	0,15	332	0,94	0,49
2.	azon.lt	36643	1	1	34	1	0,32	370	1	0,87
3.	bigbox.lt	10282	0,68	0,94	13	0,88	0,54	176	0,8	0,96
4.	citylights.com	2026	1	0,64	22	0,33	0,36	190	1	0,56
5.	currys.co.uk	2009	1	0,81	56	1	0,3	627	1	0,45
6.	elshop.lt	22019	1	1	22	1	0,55	244	1	0,76
7.	ikea.com	4909	1	1	9	1	0,56	99	1	1
8.	ilterzogirone.it	2866	1	1	10	1	0,7	49	1	0,98
9.	imk.lt	10173	0,96	1	16	0,82	0,56	191	0,95	0,93
10.	iristorante.it	967	1	0,9	4	1	1	35	1	0,74
11.	kompiutera.lt	11440	0,54	1	41	0,44	1	528	0,53	1
12.	smartbuy.lt	3632	1	1	36	1	0,25	396	1	0,99
13.	tesco.com	13066	0,97	0,96	30	0,81	0,83	274	0,79	0,15
14.	varle.lt	59450	1	1	48	1	0,73	483	0,98	0,98
	Vidurkis:	14996	0.93	0,93	27	0,88	0,56	285	0,93	0,78

Kaip matyti iš S3 lentelės, kur rodomas vykdymo laikas sekundėmis, UXClust metodas suklasterizuoja į matuojamą klasterį vidutiniškai 893 tinklalapius per sekundę, kai tuo tarpu pg-Grams atitinkama reikšmė yra tik 0,4 ir CP – tik 3,14. Be to, kaip yra žinoma, UXClust metodu buvo klasterizuojami visi kiekvienos internetinės svetainės

tinklalapiai, t. y. viso daugiau kaip vienas milijonas. Gerokai lėtesniems kitų autorių pg-Grams ir CP metodams iš kiekvienos internetinės svetainės buvo naudojama atitinkamai tik 100 ir 1000 tinklalapių. Iš absoliutaus vykdymo laiko matyti, jog UXClust metodas suklasterizavo daugiau kaip vieną milijoną tinklalapių per 235 sekundes, pg-Grams metodu suklasterizuoti 1400 tinklalapių pavyko per 83 sekundes, o su CP metodu – 14000 tinklalapių per 132 sekundes. Būtent klasterizavimo laike ir išryškėja neabejotinas UXClust metodo pranašumas.

S3 lentelė. Vykdymo laikas

Nr.	**Svetainė**	**UXClust**		**pg-Grams (100)**		**CP (1000)**	
		laikas (s)	*teisingų/s*	*laikas (s)*	*teisingų/s*	*laikas (s)*	*teisingų/s*
1.	argos.co.uk	31	982	49	0,82	61	5,44
2.	azon.lt	12	3053	66	0,52	71	5,21
3.	bigbox.lt	35	293	123	0,11	190	0,93
4.	citylights.com	1	2026	21	1,05	26	7,31
5.	currys.co.uk	1	2009	61	0,92	67	9,36
6.	elshop.lt	5	4403	16	1,38	37	6,59
7.	ikea.com	18	272	65	0,14	89	1,11
8.	ilterzogirone.it	12	238	70	0,14	38	1,29
9.	imk.lt	25	406	108	0,15	153	1,25
10.	iristorante.it	13	74	50	0,08	46	0,76
11.	kompiutera.lt	5	2288	54	0,76	100	5,28
12.	smartbuy.lt	2	1816	21	1,71	31	12,77
13.	tesco.com	56	233	165	0,18	229	1,20
14.	varle.lt	17	3497	83	0,58	132	3,66
Viso:		235	–	950	–	1271	–
Vidurkis:		–	893	–	0,40	–	3,14

ClustVX atveju buvo sudarytas vienas testinis rinkinys turintis tinklalapius su struktūrizuotais duomenimis iš dešimties skirtingo dizaino interneto svetainių, bei buvo pasinaudota dar trimis viešai prieinamais testiniais tinklalapių rinkiniais ((Zhao *et al.* 2005), (Álvarez *et al.* 2008) ir (Yamada, Craswell 2004)). Visi keturi testiniai rinkiniai buvo sudaryti iš 363 skirtingo dizaino interneto svetainių. S4 lentelėje pateikiami šių keturių testinių rinkinių duomenys.

Kai kurie testiniai rinkiniai turėjo daugiau negu vieną tinklalapį iš vienos internetinės svetainės. Vykdant eksperimentinius bandymus buvo skaičiuotas struktūrizuotų duomenų įrašų kiekis imant tik po vieną tinklalapį iš kiekvienos skirtingo dizaino svetainės.

Tiriamų metodų efektyvumas buvo skaičiuojamas remiantis dvejais rodikliais: tikslumu ir atkuriamumu. Šie rodikliai skaičiuojami remiantis skaičiumi struktūrizuotų duomenų įrašų esančių tinklalapyje ($DR_{esantys\_tinklalapyje}$), skaičiumi išgautų duomenų įrašų iš tinklalapio ($DR_{visi\_išgauti}$), skaičiumi teisingai išgautų duomenų įrašų ($DR_{teisingi}$). Šie rodikliai kiekvienam tinklalapiui yra paskaičiuojami atskirai ir rezultatų lentelėse rodomas bendras vidurkis. Minėtiems rodikliams apskaičiuoti naudojamos šios formulės:

$$Tikslumas = \frac{|DR_{teisingi} \cap DR_{visi\_išgauti}|}{|DR_{visi\_išgauti}|}, \quad (4)$$

$$Atkuriamumas = \frac{|DR_{visi\_išgauti} \cap DR_{esantys\_tinklalapyje}|}{|DR_{esantys\_tinklalapyje}|}. \quad (5)$$

S4 lentelė. Testiniai tinklalapių rinkiniai naudoti vykdant metodų eksperimentinius tyrimus

Testinis tinklalapių rinkinys:	**ClustVX**	**ViNTs-2**	**Alvarez**	**TBDW**
Svetainių	10	102	200	51
Tinklalapių iš vienos svetainės	3	11	1	5
Struktūrizuotų duomenų įrašų tinklalapyje	22	24	18	21
Viso įrašų (imant po vieną tinklalapį iš svetainės)	218	2489	3557	1052

Žemiau esančiuose lentelėse pateikiami eksperimentinių bandymų rezultatai su kiekvienu testiniu tinklalapių rinkiniu atskirai. Kaip matyti iš pateiktų rezultatų, siūlomas ClustVX metodais su visais testiniais tinklalapių rinkiniais savo efektyvumu lenkia kitų autorių metodų rezultatus.

S5 lentelė. Bandymų rezultatai naudojant ClustVX testinį tinklalapių rinkinį

Metodas / Rodiklis	**ViNTs**	**FiVaTech**	**MDR**	**ClustVX**
Tikslumas	86,0%	97,7%	50,0%	100,0%
Atkuriamumas	65,6%	59,2%	9,2%	99,5%

Pažymėtina, jog didžiausias skirtumas tarp siūlomo ClustVX metodo ir kitų autorių metodų matomas S5 lentelėje. Kaip žinia, šis testinis tinklalapių rinkinys buvo sudarytas iš šiuolaikinių technologiškai sudėtingų tinklalapių. Tokie tinklalapiai pasižymi ilgu

HTML kodu, daugybe reklamų, JavaScript kodo vykdymu, bei kitomis technologiškai sudėtingomis ypatybėmis, kurios sukelia sunkumų išgaunant struktūrizuotus duomenis naudojant kiek senesnius kitų autorių metodus.

S6 lentelė. Bandymų rezultatai naudojant VINTS-2 testinį tinklalapių rinkinį

Metodas / Rodiklis	**ClustVX**	**G-STM**	**DEPTA**
Esantys įrašai	2489	N/D	N/D
Išgauti įrašai	2452	N/D	N/D
Teisingai išgauti	2417	N/D	N/D
Tikslumas	98,6%	98,5%	95,1%
Atkuriamumas	98,5%	96,7%	83,9%

Kaip matoma iš S6 lentelės, struktūrizuotų duomenų įrašų gavybos rezultatai su VINTS-2 testiniu rinkiniu vėl atskleidė, jog siūlomas ClustVX metodas pasiekia labai aukštą tikslumą ir atkuriamumą, t. y. 98,6% ir 98,5% atitinkamai. Šie rezultatai yra geresni negu kitų dviejų metodų. Naudojant šį testinį rinkinį didžioji dalis netikslumų atsiranda, kuomet ClustVX metodas netinkamai segmentuoja struktūrizuotų duomenų įrašus tinklalapyje.

S7 lentelė. Bandymų rezultatai naudojant Alvarez testinį tinklalapių rinkinį

Metodas:	**ClustVX**	**Alvarez et al.**
Esantys įrašai	3557	3557
Išgauti įrašai	3546	3570
Teisingai išgauti	3482	3496
Tikslumas	98,2%	97,9%
Atkuriamumas	99,7%	98,3%

S7 lentelėje pateikiami bandymų rezultatai naudojant (Álvarez *et al.* 2008) sudarytą testinį tinklalapių rinkinį. Verta atkreipti dėmesį, jog pats Alvarez metodas tinklalapiuose randa ir išgauna daugiau struktūrizuotų duomenų įrašų negu jų iš tiesų yra. Tai gali būti įvardijama kaip klaidingai teigiamas (angl. *false positive*) struktūrizuotų duomenų išgavimas ir tokio tipo sprendimai mažina metodo tikslumą.

Iš S8 lentelėje pateiktų duomenų matyti, jog siūlomas ClustVX metodas vėlgi savo tikslumu ir atkuriamumu nenusileidžia kitų autorių metodams. Pažymėtina, jog ClustVX metodas pasiekia absoliučiai geriausią tikslumą, kuris lygus net 99,5%. Maža dalis klaidų, kurios neleido pasiekti šimtaprocentinio tikslumo, buvo susijusios su tuo, jog struktūrizuotų duomenų įrašų lentelės pirmoji antraštinė eilutė buvo priskiriamas kaip atskiras duomenų įrašas. Šio tipo klaidų gana sudėtinga išvengti, nes lentelės antraštinė

eilutė dažnai yra vizualiai ir struktūriškai labai panaši į duomenų įrašus ir skirtumą galima pastebėti tik atliekant pateikto teksto semantinę analizę.

S8 lentelė. Bandymų rezultatai naudojant TBDW testinį tinklalapių rinkinį

Metodas:	**ClustVX**	**G-STM**	**DEPTA**	**FiVa Tech**	**CTVS**	**DeLa**
Esantys įrašai	1052	N/A	N/A	693	693	693
Išgauti įrašai	1047	N/A	N/A	690	688	655
Teisingai išgauti	1045	N/A	N/A	672	680	616
Tikslumas	99,8%	99,8%	99,5%	97,0%	98,8%	88,8%
Atkuriamumas	99,5%	96,6%	85,3%	97,4%	98,1%	94,0%

Taigi, šiame skyriuje eksperimentiniais tyrimais buvo išbandyti abu siūlomi metodai: ClustVX ir UXClust. Bandymų rezultatai atskleidė, kad abu siūlomi metodai savo efektyvumu, kuris skaičiuojamas kaip tikslumas ir atkuriamumas, lenkia daugelį kitų autorių metodų.

Bendrosios išvados

1. Literatūros analizė atskleidė, kad rankiniu žmogaus darbu grįsti struktūrizuotų duomenų išgavimo metodai reikalauja didelių kaštų ir kartu nėra tinkami išgauti duomenis interneto mastu, t. y. iš tūkstančių vizualiai ir struktūriškai skirtingų tinklalapių. Todėl dauguma šiuolaikinių automatinių duomenų išgavimo metodų tinklalapiuose, sugeneruotuose pagal šablonus, ieško struktūrinių pasikartojimų, kuriuos išnaudoja automatiškai identifikuojant ir išgaunant struktūrizuotus duomenis. Tačiau visiškai automatinis interneto masto struktūrizuotas duomenų išgavimas, kurio tikslumas ir atkuriamumas būtų pakankamas praktinėms problemoms spręsti, vis dar išlieka iki šiol nepasiektas tikslas.
2. Pagal šablonus sugeneruoti tinklalapiai turi struktūriškai ir vizualiai pasikartojančių elementų. Disertacijoje siūloma nauja vizualiai ir struktūriškai panašių tinklalapių klasterizavimo technika padeda surasti šiuos struktūrinius pasikartojimus ir juos suklasterizuoti. Gauti klasteriai su tinklalapio elementų XPath adresais gali būti išnaudojami automatiškai generuojant iš tinklalapių struktūrizuotus duomenis išgaunančias taisykles.
3. Eksperimentiniais tyrimais parodyta, kad siūlomas ClustVX metodas, skirtas struktūrizuotų duomenų išgavimui, pasiekia 98% tikslumą bei atkuriamumą ir tuo lenkia visus kitus bandymuose palyginimui naudotus

šiuolaikinius struktūrizuotų duomenų išgavimo metodus. Be to, siūlomas metodas gali išgauti duomenis iš technologiškai sudėtingų tinklalapių.

4. Pasinaudojant internetinės svetainės vidinių nuorodų XPath adresais tinklalapiuose galima ženkliai pagreitinti tinklalapių klasterizavimą pagal jų struktūrinį panašumą. Be to, parodyta, kad klasterizavimo metu skaičiuojant struktūrinį panašumą tarp tinklalapių, galima naudoti Žakaro (Jaccard) panašumo tarp baigtinių rinkinių koeficientą.
5. Pasiūlytas UXClust metodas, skirtas sparčiam tinklalapių klasterizavimui, yra daugiau kaip 200 kartų spartesnis negu lyginimui pasirinkti kiti du šiuolaikiniai kitų autorių metodai. Pasiūlytas metodas yra pajėgus suklasterizuoti daugiau kaip 1 milijoną tinklalapių per nepilnas 4 minutes ir kartu išlaikyti didesnį negu 90% tikslumą ir atkuriamumą.
6. Abu pasiūlyti metodai, skirti struktūrizuotų duomenų išgavimui ir tinklalapių klasterizavimui pagal struktūrinį panašumą, nereikalauja išankstinio apmokymo, veikia automatiškai, yra nejautrūs teminėms sritims, todėl gali būti panaudoti kuriant interneto masto struktūrizuotų duomenų išgavimo sistemas.

Annexes[13]

Annex A. The Co-authors Agreements to Present Publications for the Dissertation Defence
Annex B. Copies of Scientific Publications by the Author on the Topic of the Dissertation

[13] The annexes are supplied in the enclosed compact disc

STRUCTURED DATA EXTRACTION FROM TEMPLATE-GENERATED WEB PAGES

Doctoral Dissertation

Technological Sciences,
Informatics Engineering (07T)

Tomas Grigalis

STRUKTŪRIZUOTŲ DUOMENŲ IŠGAVIMAS IŠ TINKLALAPIŲ SUGENERUOTŲ PAGAL ŠABLONUS

Daktaro disertacija

Technologijos mokslai,
informatikos inžinerija (07T)

2014 08 14. 11,5 sp. l. Tiražas 20 egz.
Vilniaus Gedimino technikos universiteto
leidykla „Technika“,
Saulėtekio al. 11, 10223 Vilnius,
http://leidykla.vgtu.lt
Spausdino UAB „Baltijos kopija“
Kareivių g. 13B, 09109 Vilnius
www.kopija.lt