GEN-X2: Generation of XPath Expressions for Unsupervised Web Data Extraction

Hugo Miguel Laranjeira Guimarães

Thesis to obtain the Master of Science Degree in Information Systems and Computer Engineering

Examination Committee

Chairperson: Prof. José Carlos Martins Delgado
Supervisor: Prof. Pável Pereira Calado
Member of the Committee: Prof. José Alberto Rodrigues Pereira Sardinha

November 2013
Acknowledgments

First I would like to thank my advisor Prof. Pável Calado for all the discussions and ideas that emerged from them. Also, for understanding all the setbacks which occurred during the development of this thesis.

I have to thank my colleagues and friends Diogo Jesus and Maria João Pereira which let me be part of the best work group that anyone can have during university and that allowed me to be here today.

I would also like to thank my friends Júlia Pereira, Marta Brito, Joana Branco and António Amaral for making me company while I was working on the thesis and who listened to my complaints, doubts and indirectly made me rethink about how I could improve GEN-X2.

With this final work I can finally fulfill my dream of going to work abroad and start a new life with my girlfriend Katarzyna Sekulska. Thank you for the final extra pressure to finish and deliver all the knowledge I got throughout the development of GEN-X2.

And last, but definitely not least, I must thank my parents and my sisters for their unconditional support and confidence in my abilities, in my work and in my creativity, not only during the thesis, as well as throughout my academic career.

Lisboa, October 2013

Hugo Guimarães
Abstract

The Internet is widely used by everyone and it is possible to find almost everything online. However, there is a lot of information of interest that is not easy to access using search engines. This information comes from back-end databases and the web pages are generated dynamically after a user's request. These web pages compose the hidden web and are of great importance since the containing information can be used to make applications that compare products, flights, etc. The objective of this work is to be able to extract all the information of interest automatically and don’t extract the information that is not of our interest, like ads or "left bars", without any human intervention. There are already many systems that are able to do this, however they all have problems and they are not completely efficient performing the task. This dissertation presents a novel approach to extract and store information in an unsupervised manner, using Genetic Programming and XPath as resources to reach a solution. Our system, called GEN-X2, achieved a precision of 74.66% and recall of 76.06%.

Keywords: Unsupervised information extraction, Visual information, Hidden web, Genetic programming, XPath
Resumo

A Internet é amplamente utilizada por todos e é possível encontrar quase tudo online. No entanto, existe uma grande quantidade de informação de interesse que não é de fácil acesso utilizando motores de busca. Essa informação está armazenada em bancos de dados e é mostrada a partir de páginas que são geradas dinamicamente a pedido de um usuário. Estas páginas compõem a hidden web e são de grande importância, pois a informação contida nas mesmas pode ser utilizada para criar aplicações que comparam produtos, voos, etc. O objetivo deste trabalho é de ser capaz de extrair todas as informações de interesse automaticamente e não extrair a informação que não é do nosso interesse, como anúncios ou barras laterais, sem qualquer intervenção humana. Existem já diversos sistemas que são capazes de o fazer, mas todos eles têm problemas e não são completamente eficientes a realizar a tarefa. Este relatório apresenta uma nova abordagem para extrair e armazenar informação de forma não supervisionada, usando Programação Genética e XPath como recursos para chegar a uma solução. O nosso sistema, chamado GEN-X2, obteve uma precisão de 74.66% e recall de 76.06%.

Keywords: Extração de informação não supervisionada, Informação visual, Hidden web, Programação Genética, XPath
Contents

Acknowledgments

Abstract

Resumo

1 Introduction

1.1 Motivation

1.2 Thesis Proposal

1.3 Contributions

1.4 Dissertation Outline

1.5 Summary

2 Concepts

2.1 Web Data Structure

2.1.1 Web Pages

2.1.2 XPath

2.2 Web Data Extraction

2.3 Genetic Programming

2.3.1 Grammatical Evolution

2.4 Named-Entity Recognition
4.2.3 Grammar Creator ................................................. 45
4.2.4 GEVA .......................................................... 46
4.2.5 Labeler .......................................................... 46

4.3 Fitness Function .................................................... 47
4.3.1 Feature 1 - Centered information is better ................. 47
4.3.2 Feature 2 - Big area is better ............................... 48
4.3.3 Feature 3 - More characters is better ...................... 50
4.3.4 Feature 4 - No area intersection is better ............... 50
4.3.5 Feature 5 - Subexpressions must have only one type .... 50
4.3.6 Feature 6 - Less subexpressions is better ............... 51
4.3.7 Feature 7 - Deep subexpressions is better ............... 52
4.3.8 Feature 8 - Expression possible in more web pages is better ... 52
4.3.9 Feature 9 - Distinct tokens are better .................... 52
4.3.10 Feature 10 - More subexpressions possible is better .... 53
4.3.11 Feature 11 - Subexpressions must be different ......... 53
4.3.12 Feature 12 - Subexpressions have a possible label .... 54

4.4 Summary .......................................................... 54

5 Validation and Experimental Results .......................... 55
5.1 Validation .......................................................... 55
5.1.1 Datasets used in the Experiments .......................... 56
5.2 Results and Discussion ........................................... 56
5.2.1 Performance .................................................... 56
5.2.2 General Performance ......................................... 59
5.3 Summary .......................................................... 60

6 Conclusions and Future Work ..................................... 61
## List of Tables

<table>
<thead>
<tr>
<th>Table Number</th>
<th>Table Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Examples of the result of applying a XPath Expression</td>
<td>11</td>
</tr>
<tr>
<td>3.2</td>
<td>Unsupervised extraction systems characteristics comparison</td>
<td>39</td>
</tr>
<tr>
<td>3.3</td>
<td>Unsupervised extraction systems tasks performed comparison</td>
<td>40</td>
</tr>
<tr>
<td>A.4</td>
<td>GEVA input parameters</td>
<td>69</td>
</tr>
<tr>
<td>B.5</td>
<td>General Performance in GEN-X2</td>
<td>72</td>
</tr>
</tbody>
</table>
List of Figures

1.1 Example of a web page with a Search Query Interface ................. 2
1.2 Example of a Search Results web page .............................. 3
2.3 Bookstore XML example ........................................ 10
2.4 Genetic Programming overview. Adapted from Koza (1992) ........... 14
3.5 RoadRunner execution example (Crescenzi et al. (2001)) ............. 20
3.6 Iteratively discovering C-repeated patterns (Wang & Lochovsky (2003)) . 22
3.7 Partial Alignment - Inserting nodes in the seed tree: (A) and (B) Unique expansion; (C) Insertion ambiguity. (Zhai & Liu (2005)) .......... 25
3.8 ViDE Visual Block tree example Liu et al. (2010) ......................... 28
3.9 ViDE data items alignment example (Liu et al. (2010)) ............... 29
3.10 Block and correspondent shape example (Zhao & Meng (2005)) ...... 30
3.11 Visual histogram (Simon & Lausen (2005)) ............................. 32
4.12 XPath expression and subexpressions example ....................... 42
4.13 Redundant XPath subexpressions .................................. 42
4.14 System architecture .............................................. 43
4.15 Line sweep algorithm example .................................... 49
5.16 GEN-X2 precision and recall ...................................... 57
5.17 GEN-X2 general performance ..................................... 58
5.18 Extraction example .............................................. 59
Chapter 1

Introduction

Information is being produced like never before and tends to increase. According to Gantz & Reinsel (2011), the digital universe has about 1.8 trillion gigabytes and it is more than doubling every two years. More than time, information is money and the key is to get the right one and ignore useless information. Gantz & Reinsel (2012) reported that besides the great amount of information available only 23% might be useful if tagged and analyzed. However, merely 0.5% is actually being analyzed.

Web data extraction appeared to retrieve and organize the existing information available online. Several approaches have been created in order to maximize the information picked up correctly, and to minimize the search time, as cited in Laender et al. (2002) and Chang et al. (2006).

An ideal solution would be to get all the valuable information and ignore the useless one in an automatic manner without any human interference, making it available at any time someone needs it. Our proposed system, GEN-X2, gives some steps in that direction by using Genetic Programming (GP) and XPath expressions.

1.1 Motivation

The Internet is widely used by everyone and it is possible to find almost everything online. Terms like google it became normal in daily conversations and searching for someone
in social networks became an usual activity. Nonetheless, there is a lot of information that is not easily reachable, unless we know where and how to find it. The main reasons why this happens are the tremendous amount of information available online and the existence of two types of web. There is the surface web, which is composed with static web pages, and there is the deep web, made of dynamically generated web pages, that shows information stored in databases in a more appealing way to users. There are surveys that say the deep web is 500 times larger than the surface web as stated by Chang et al. (2004).

As a result, though indirectly related, there are two main ways to search information online. First, to search using search engines, writing down some keywords related to the topic we want to find information about. Second, to search using web pages search query interface. In Figure 1.1 we can see an example of a search query interface on the left of the web page which enables a user to query the hidden database of this company to see the available flights.

The goal is to automatically be able to get the information from deep web pages and to do this there are four main tasks that must be performed. First, to identify the region of the web page that has the information of interest. Second, to identify the objects of interest in the region identified before. Third, to align the attributes of the objects identified before and finally to label these attributes.

In Figure 1.2 we can see a different web page that shows the results of a query made by
1.1. MOTIVATION

A user that wants to travel from Lisbon to Vienna. Each of the rows is a data record and each column is a data item. In the second row we have a data record composed by two flights. So we can figure out that each flight is considered a data item of the data record which permits to go from Lisbon to Vienna.

![Image of a flight booking page]

**Figure 1.2:** Example of a Search Results web page

The biggest problem of extracting information automatically is the enormous ways that a web designer has to develop a web page and automatic information extraction systems have to make a lot of assumptions in order to do it. However, web designers present the information in an easy way to human users. Identifying optional and disjunctive attributes is also problematic.

With this thesis, a novel approach which combines Genetic Programming (GP) and XPath is proposed in order to retrieve web information using GP to evolve and rate the XPath that are able to extract the information.
1.2 Thesis Proposal

A new approach is revealed for the task of automated web data extraction using Genetic Programming in order to find the best XPath expression to retrieve useful information. Textual and visual information are combined to rank XPath expressions and a grammar evolution tool, GEVA (O’Neill et al. (2008)), is used to evolve a group of XPath expressions and return the fittest one.

Since a complete dataset was not available according to our criteria of acceptance, two experiments were conducted in order to validate our approach, called GEN-X2. The first evaluates the precision and recall according to the chosen dataset. The second is a more subjective approach which verifies if the XPath expressions produced retrieve information correctly according to a looser criteria evaluating the extracted information into three different categories.

1.3 Contributions

The main contributions of this work can be summarized as follows:

**GEN-X2:** the system developed during this thesis implements a new approach to the problem of web data extraction, i.e., it unifies Genetic Programming and XPath expressions as a mean to retrieve information in an automatic manner from general web pages. HTML web pages are cleaned in order to be parsed by Java DOM parser correcting some possible existing errors. The web page is rendered and the position of each HTML element is added as attributes of the element. A BNF grammar is created by the collection of the web pages given as input so XPath expressions can be generated through the GEVA (the grammatical evolution tool). After that GEVA returns the best XPATH expression found during the run. The system is available in my Bitbucket account \(^\text{1}\).

\(^{1}\)https://bitbucket.org/HugoGuimaraes
1.4. DISSERTATION OUTLINE

**Set of Features to evaluate XPath expressions** : ranks the XPath expressions according to the likelihood of containing relevant information. It can work as a ranking system, since it is able to characterize XPath expressions using visual (position, area), textual (quantity of text and repetitions) and self-contained (number of subexpressions and depth) features, which can be combined in endless ways.

**Utils Library** : that attempts to simplify the creation of BNF grammars from XHTML web pages (single or collection) and the acquisition of each HTML element position in the web page through the renderization of it. These were the most laborious tasks with fewer information and examples online, so we think it can be a great help to future users of this approach and it can be useful for other areas besides web data extraction as well. It is also available in my Bitbucket account.

### 1.4 Dissertation Outline

The contents of this dissertation are organized as follows:

Chapter 2 presents basic, but important, concepts which are critical for understanding this work.

Chapter 3 gives a general idea of the state of the art in unsupervised web data extraction dividing each work in smaller tasks to understand and take advantage of the knowledge produced by them.

Chapter 4 describes GEN-X2, our proposed system, in detail giving insight over the architecture and the way XPath expressions are rated using textual, visual and self-contained information.

Chapter 5 describes the methods used to evaluate and validate the system.

Finally, Chapter 6 completes the work and discusses possible future work to enhance the system performance.
1.5 Summary

Chapter 1 delineates the motivation behind this dissertation and explains why unsupervised web data extraction is so important. An overview of the system created is provided and the contributions resulting from this work are enumerated. Likewise, the following Chapters are described.
Chapter 2

Concepts

This Chapter presents the fundamental concepts required for understanding this dissertation. There are three main areas: it explains how web pages in the Internet are generally displayed, it presents some basic concepts and it gives insights over the paradigm of Genetic Programming.

2.1 Web Data Structure

This section introduces some concepts about how information is displayed in web pages and how is possible to reach it.

2.1.1 Web Pages

Almost every website with information of interest has some structure. The main reasons for that are the items presented in those web pages come from a hidden database and the corresponding web pages are created automatically using a script that gets the items and displays them in a structured manner. These web pages that are created by the same script are known as collection of web pages (Crescenzi et al. (2001)). The reason for not having completely unstructured websites is because it is easier for a user to understand a structured web page than a completely unstructured one.

Concerning to pages from the Hidden Web, there are two types of pages:
List pages: displays a list of one or more objects, without many detailed information.

Detail pages: give all the information about a specific object.

To understand how the information is presented in a web page it is important to understand the concepts of data region and data record.

Data region: is the area of a web page where the information we want to retrieve is. It can be seen as a group of data records.

Data record: all the information related to one object.

Data item: each attribute of a data record.

In Figure 1.2 there is a data region formed by the rows and columns, three data records and each cell of the table is a data item. It is important to notice that in the second data record there are two data items of the type flight.

A data record can be: (i) contiguous when it is altogether or (ii) non-contiguous when it is not altogether, i.e., considering the following sequence part1 of object1, part1 of object2, part2 of object1, part2 of object2, we can see that object1 and object2 are non-contiguous (Liu et al. (2003)).

In order to generate wrappers, i.e, the program that extracts data from web pages, it is necessary to get the web pages and there are two possibilities to do that. The first one is not very efficient and it is time consuming because you have to do it manually. A better possibility is to use a web crawler that do this work automatically. A web crawler sends queries to the web site and gets the resulting web pages and possible labels (Wang & Lochovsky (2003)).

2.1.2 XPath

For understanding why XML Path Language ¹ (XPath) is important for this work, it is necessary to perceive the relation between eXtensible Markup Language ² (XML) and

¹http://www.w3schools.com/xpath/xpath_intro.asp
²http://www.w3schools.com/xml/xml_whatis.asp
HyperText Markup Language\(^1\) (HTML).

XML was created to be both human-readable and machine-readable and having as a purpose to carry data instead of displaying it. The language is not predefined, making it possible to add new tags. An example of an XML document representing a bookstore is presented in Figure 2.3(a).

Most of web pages are coded in HTML because it describes presentation instead of content, being human-readable after being rendered by a web browser.

In this work, the HTML web pages will be converted to EXtensible HyperText Markup Language\(^2\) (XHTML) because it is a stricter and cleaner version of HTML and, in fact, it represents HTML defined as an XML application. The most important differences between XHTML and HTML are: (i) all elements and attribute names must be lowercase, (ii) all empty elements must be close and (iii) all attribute values must be quoted.

Finally, XPath is a programming language that allow us to query XML documents. Considering an XML document as a tree, XPath permits to select nodes imposing conditions. The same can be applied to an HTML or XHTML document since they have a more restrict grammar but have similar structure to XML documents. An example of an XML document represented as a tree is presented in Figure 2.3(b).

In this dissertation, XPath is of main importance because it will give us a mean to access nodes of the XHTML documents.

In Table 2.1, we can see some examples of how XPath expressions can be used.

All of these languages are a W3C Recommendation\(^3\) which have as a mission to lead the World Wide Web to its full potential by developing protocols and guidelines that ensure the long-term growth of the Web. This means that being a W3C Recommendation, these languages have more probability of continue being the most used online.

\(^{1}\)http://www.w3schools.com/html/html_intro.asp

\(^{2}\)http://www.w3schools.com/html/html_xhtml.asp

\(^{3}\)http://www.w3.org/
2.2 Web Data Extraction

This section will give an overview about web data extraction (WDE), which focus in extraction information from web pages. The following concepts are of importance:

**Wrappers or extractors**: Computer programs that are able to extract data records from web pages.

**Wrapper induction or information extraction systems**: Software tools which are built to generate wrappers.

To extract important information of websites, most of web data extractors try to perform four tasks accurately. It needs to identify the data region, then to identify the data records
2.2. WEB DATA EXTRACTION

Table 2.1: Examples of the result of applying a XPath Expression

<table>
<thead>
<tr>
<th>XPath Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>/bookstore</td>
<td>Selects the root element bookstore.</td>
</tr>
<tr>
<td>//book</td>
<td>Selects all book elements no matter where they are in the document.</td>
</tr>
<tr>
<td>//@lang</td>
<td>Selects all attributes that are named lang.</td>
</tr>
<tr>
<td>//title</td>
<td>//year</td>
</tr>
<tr>
<td>descendant::book</td>
<td>Selects all book descendants of the current node.</td>
</tr>
<tr>
<td>ancestor::book</td>
<td>Selects all book ancestors of the current node.</td>
</tr>
<tr>
<td>child::*/child::price</td>
<td>Selects all price grandchildren of the current node.</td>
</tr>
</tbody>
</table>

within the data region, later to align the data items and finally label them. There are four ways to do this and every one of them has advantages and disadvantages.

**Manually constructed WDE Systems**: Users have to program a wrapper for each website. It is time consuming and the user has to be highly skilled, having computer and programming knowledge. Although, it is the best option for web pages that don’t change very often. For example: TSIMMIS (Hammer et al. (1997)), Minerva (Crescenzi & Mecca (1998)), WebOQL (Arocena & Mendelzon (1998)), W4F (Sahuguet & Azavant (2000)) and XWrap (Liu et al. (2000)).

**Supervised WDE Systems**: The system learns how to extract the information from completely labeled examples given by the user. User can be trained to know how to give examples to the system. For instance: SRV (Freitag (1998)), RAPIER (For et al. (1999)), WIEN (Kushmerick (1997)), WHISK (Soderland et al. (1999)), NoDoSE (Adelberg (1998)), SoftMealy (Hsu & Dung (1998)), STALKER (Muslea et al. (1999)) and DEByE (?) and (Ribeiro-Neto et al. (1999)).

**Semisupervised WDE Systems**: The system learns how to extract the information from non labeled examples given by the user. The user has the post effort of choosing the target pattern and indicate the data to be extracted. For example: IEPAD (Chang & Lui (2001)), OLERA (Chang & Kuo (2004)) and Thresher (Hogue & Karger (2005)).

**Unsupervised WDE Systems**: The system doesn’t need any labeled examples and no
user interactions to generate the wrapper. Examples are presented in the following chapter.

Since it is not essential for the understanding of this report, the first three types of systems are just referred but not deepened, whereas unsupervised systems are presented in detail in Chapter 3.

### 2.3 Genetic Programming

In this Section, an overview is presented of what Genetic Programming (GP) is. GP is a methodology that allow us to create a computer program from existing ones (Poli et al. (2008)). The user doesn’t need to have any knowledge about the solution in advance. Having just an idea of the goal we want to achieve, the computer program evolves to the desired result or at least a closer solution. The returned computer program is the best possibility to solve the problem. In this paradigm, computer programs are usually seen as syntax trees. To build each tree (program) some basic components are needed:

**Terminals**: Leaves of the tree. It can be the program’s external inputs, functions with no arguments or constants.

**Functions**: Internal nodes of the tree.

**Primitive set**: Set with combinations of functions and terminals allowed.

To get the best computer program a set of computer programs and a way of evaluating them in order to select the fittest one is necessary. To be able to get the computer program that is the better for the problem we have in hands, various steps must be performed for several iterations.

**Population**: Number of computer programs built in each iteration.

**Fitness function**: Function that gives a score to each computer program.
Selection: The way we pick up the computer programs to become a parent in the next iteration. There are several ways of doing this, but the most common one is Tournament Selection that chooses randomly some computer programs and compares them using the result of the fitness function. The ones with the best result have better chances to be chosen as a parent.

The fitness function is from main importance, since it is the one that allow us to evaluate the performance of each program. A good fitness function has more probability of giving us the right solution and possibly in less iterations. As a first step, an initial population is created and there are three common ways to do it:

Full: Every leaf is at the same depth. To do that we choose functions until the last level and then we choose just terminals.

Growth: Leaves can be at different depths. To do that we choose terminals or functions for every level until the last one. Then, we can only choose terminals.

Ramped-half-and-half: Combination of full and growth methods. This is done using several different depth limits to make sure that we generate trees having a variety of sizes and shapes.

After preparing the first generation of computer programs, and after evaluating and selecting the fittest ones, we need to have ways of combining them. To do that we have two types:

Mutation: We use only one parent program to generate a child one, simply changing some part of the parent.

Crossover: We use two parent programs to generate a child one, choosing and combining parts of the two.

Reproduction: Returns a copy of a parent program.

In a nutshell, as we can see in Figure 2.4, the first step is to create an initial population of programs, then we need to evaluate if the termination criterion is still not satisfied. If yes
Figure 2.4: Genetic Programming overview. Adapted from Koza (1992)

we return the result, if not we need to evaluate each individual of the population using the fitness function. Until the following generation is created we create a new individual using one of the three possibilities: mutation, crossover or reproduction. The choice is made using probabilities. To create an individual using mutation, one individual from the previous generation is selected and a transformation is made in order to generate a new individual. To create an individual using crossover, two individuals from the previous generation are selected and they are combined in order to generate a new one. To create an individual using reproduction, one individual from the previous generation is selected and copied. It ends when the termination criterion is satisfied returning the best individual.

It is not normal to return a result just after creating the initial population because if that happens, Genetic Programming is not necessarily needed. The selection of individuals is
made by combining the result of the fitness function with a certain probability, i.e., higher scores corresponds to superior probabilities and so it is more probable to be selected.

2.3.1 Grammatical Evolution

As Genetic Programming, Grammatical Evolution is able to find solutions to problems where a solution is not available at the beginning. It uses genetic operators, as crossover, mutation and reproduction. The only difference is that the individuals (initial population and following ones) are built under a grammar constraint. As cited in O’Neill et al. (2008), it is a grammar-based form of Genetic Programming. The grammar is usually written in Backus-Naur form (BNF grammar). A simple example of a BNF grammar is presented in Listing 2.1.

| <Exp> ::= <Var> | (0) |
| <Exp> ::= <Exp> <Op> <Exp> | (1) |
| <Var> ::= x | (0) |
| <Op> ::= + | (0) |
| * | (1) |
| − | (2) |
| / | (3) |

Listing 2.1: BNF grammar example

In this approach, each individual is composed by a genotype (list of integers which encode the rules to choose from the grammar) and a phenotype (the actual correspondence). For example, for a genotype 100300 the correspondent phenotype is x / x. First, <Exp><Op><Exp> is selected. Then <Var> is selected for the first <Exp> and with the second 0, x is attributed to <Var>. With 3, / is attributed to <Op>. And for the final <Exp> the same process occurs for the attribution of the second x.

It is important to note that genotypes are bigger than the correspondent phenotype and to use genetic operations, it is only needed to make changes in a list of integers and then map the correspondent phenotype to see the actual result of the change.
2.4 Named-Entity Recognition

Named-Entity Recognition (NER) is considered a subtask of information extraction which tries to correctly identify elements into predefined categories. The categories can be simple ones, like numbers, monetary values, percentages, expressions of times, etc... or more complicated, like names of people, organizations, locations. This is a big problem because the same name can represent two completely different entities. For example: Deco, depending on the context we might me talking about the football player or the Portuguese Association for Consumer Protection.

For this purpose we studied Freebase \(^1\) and DBpedia \(^2\). Besides of being able to download the data that comprises both of them, Freebase also offers a public API in order to access the data more easily and faster than processing the entire datasets by ourselves.

2.4.1 Freebase

Freebase was created with the intention of becoming a public repository of human knowledge scalable and is composed of diverse and heterogeneous information Bollacker et al. (2008). It merges the scalability of structural databases with the diversity of collaborative wikis. It is a good option for this system because it is intended to be perpetually available service. It is accessible through an API with big rate limit (100,000 per day at the day of "inscription").

2.5 Summary

In this Chapter, general characteristics of web pages are described and how XPath could be used to help in a web data extraction context. Likewise, the four main approaches to web data extraction are presented and the main concepts of Genetic Programming and consequently Grammatical Evolution are detailed.

\(^1\)http://www.freebase.com/
\(^2\)http://dbpedia.org/About
Chapter 3

Related Work

In this Chapter, we will present some of the work that is already been made in this area. To be able to extract the information from web pages in an automatic way, first we need to be able to select the area where we want to extract the information. Then, we need to be able to extract the right information. After that, we need to align the data, and finally we should be able to label our data. This section groups the systems into three categories, explaining each system in general and gives special attention to the heuristics and assumptions made by the authors in order to develop the systems. This is the information that is more useful for developing a new system with the best of each approach. To present each system, we explain how they perform each of the tasks inherent to an unsupervised WDE system. Namely, identify data region, data segmentation, data alignment and data labeling. However, some systems do not play all the tasks and as such, we present only the tasks they perform with sufficient detail to understand this thesis.

3.1 Text-Based and Tag-Structure-Based Approach

In this section, we will introduce some WDE systems that perform their tasks by analyzing the structure of tags and the text inside the tags.
3.1.1 Mining Data Records in Web pages

Mining Data Records in Web pages (MDR) is able to find data records formed by HTML tags related to table and form tags (Liu et al. (2003)). It transforms the HTML tags into a string and uses string matching algorithms to compare them in order to find data regions and therefore the data records. MDR is able to retrieve contiguous and non-contiguous data records and to find more than one data region if there is at least two or more data records in each data region. This is possible because it assumes that it is possible to have more than one data region for each page.

First, the algorithm builds a HTML tag tree of the page in question in which every pair of tags is one node. Then, it finds out the existing data regions and finally the data records in each data region.

Identify data region

The authors of MDR introduce the concept of generalized nodes. They are nodes that have the same parent node and are adjacent. It is important because in MDR, a data region is formed by at least two generalized nodes that have the same parent node, the same number of children nodes, are adjacent and the string comparison between them is lower than a given threshold. The value for the threshold is obtained using a set of training pages.

It identifies all the possible data regions and then chooses the parent level data regions that cover the child level data regions.

Data segmentation

A generalized node may not represent a data record. To find data records from each data region, MDR assumes that if a data region contains more than one data record, then they must have similar tag strings.
3.1.2 RoadRunner

RoadRunner finds similarities and differences between web pages belonging to a collection (Crescenzi et al. (2001)). Initially, a web page is chosen as the initial wrapper. Then, at each step it tries to match the wrapper produced in the previous step to another web page (sample), generalizing the wrapper in case of mismatches. The mismatches identify the relevant information. In the end more than a wrapper may be produced because it doesn’t assume that all the pages can be described using only one. The input HTML web pages are transformed to XHTML and a DOM tree is built using JTidy\(^1\).

**Identify data region and Data segmentation**

The two steps are done at the same time where the result is a grammar (wrapper) that identifies the data records. To do it, mismatches have to be threatened. There are two types of mismatches: the string mismatch and the tag mismatch.

As it can be seen in Figure 3.5 in line 4, string mismatches happen when different strings occur in correspondent positions of the wrapper and the sample. They are used to find out fields with variable values. To solve them, we simply generalize the wrapper, i.e., if previously the wrapper contained one expression with a fixed value, this one is replaced by \#PCDATA.

Tag mismatches happen when different tags occur in correspondent positions of the wrapper and the sample or between a tag and a string. There are two possibilities, the field can be (i) optional or (ii) a list (iterator). First, the algorithm checks if it is an iterator situation and if it is not the case, it checks for an optional one. If it was the other way around, it would be always possible to find optional fields, but the existence of lists would be ignored.

An iterator mismatch is verified when a list in the wrapper and in the sample have different cardinalities. As it can be seen in Figure 3.5 in line 19/20, the existence of two books in the wrapper and three in the sample. To solve it, it is necessary to identify repeated patterns (squares) and generalize the wrapper replacing the repeated pattern in the wrapper by (<RepeatedPattern>\>)+.

\(^1\)http://sourceforge.net/projects/jtidy
When it is not an iterator, an optional field may be found doing a cross-search and the wrapper is generalized by replacing the optional field by \(<\text{OptionalField}>\)?. As it can be seen in Figure 3.5 in line 6.

**Data alignment**

The data alignment is done using the grammar built in the previous step, in which the \#PCDATA fields identify the information to be withdrawn.

Data Extraction and Label Assignment for Web Databases (DeLa) uses web sites with an HTML search form to extract data records (Wang & Lochovsky (2003)). First, DeLa uses JTidy, a library for HTML cleaning, and a web crawler (HiWe) to collect web pages automatically and labels (Raghavan & Garcia-Molina (2001)). The following steps are

---

1|^http://sourceforge.net/projects/jtidy|
described below and the output is a regular expression that is able to extract information from the web pages.

**Identify data region**

DeLa uses a the Data-rich Section Extraction algorithm (DSE) to eliminate noisy data, such as advertisements and navigational panels. It needs two pages and the trees are transverse in a depth-first order, comparing all the nodes and discarding the ones that have identical sub-trees and are at the same depth. The main idea is to find C-repeated patterns, i.e., a repeated substring of a string that have at least one pair of its occurrences adjacent. In this case, this is an iterative process that is able to find the patterns in a web page. If the discovery of a pattern is made in a posterior iteration, than this pattern is dependent of patterns discovered in previous iterations. If different patterns are found in the same iterations, they are independent.

Figure 3.6 shows two iterations in which a token sequence and its correspondent token suffix-tree are represented. In Figure 6 (a), DeLa finds the C-repeated pattern "<A>text</A>". Secondly, this occurrence is masked and a new sequence is formed in Figure 6 (b) and a suffix-tree is built for the new sequence. A new C-repeated pattern "<P><A>text</A>text</P>" is found. Lastly, a wrapper can be obtained from these two iterations, "<P>(<A>text</A>)*text</P>" that correctly represents the structure for the two data objects in the given string.

**Data segmentation**

To deal with optional attributes, DeLa uses the most generalized wrapper possible. It uses a string alignment algorithm to generalize wrappers. It is chosen the generalized wrapper that can discover information from the largest number of web pages.

**Data alignment**

DeLa deals with the problem of having more than one attribute that are not separated by HTML tags, rearranging multiple values of an attribute to several rows and more than one attribute in the same string into several columns.
CHAPTER 3. RELATED WORK

Figure 3.6: Iteratively discovering C-repeated patterns (Wang & Lochovsky (2003))

Data labeling

It uses the form elements and data formats to apply a label. It uses four heuristics.

1. Match form element labels to data attributes
2. Search for voluntary labels in table headers
3. Search for voluntary labels encoded together with data attributes
4. Label data attributes in conventional formats

The web page that has the search form gives us possible labels for our data, since it has
3.2. VISUAL-INFORMATION-BASED APPROACH

to say which attributes of an item it needs to do the search. In this way, using the form elements that HiWe collected it is possible to assign a label if the keywords used to query mostly appears in a particular table column. Second, it tries to find labels in the HTML code, considering that tags related to tables usually have labels that correspond to the contained data objects. Third, they try to label using maximal-prefix and maximal-suffix within the data attributes shared by all the cells of a column. Fourth, it labels columns analyzing the data type within the column. For example, if all the cells of a column contains the symbol "@" it is probable that this cell corresponds to e-mails.

After this process, users are allowed to label the unassigned columns and to modify the labels found automatically by the process.

3.2 Visual-Information-Based Approach

In this section, we will introduce unsupervised WDE systems with a broader perspective about the web pages because they don’t use only the tag structure but also visual information. This kind of information is very important because it is what the user sees when he enters the web site and what gives him the clues about what is important or not, i.e., what is a data record and what is not. The visual information is not in the HTML code and this approach appeared in order to solve the problem of when we are just analyzing HTML code, it doesn’t contain all the information of the web page. Languages like CSS, XSLT can give a completely different look to the HTML page, possibly giving more information that can improve results if we take this into account.

3.2.1 Data Extraction based on Partial Tree Alignment

Data Extraction based on Partial Tree Alignment (DEPTA) is an improvement of MDR (Zhai & Liu (2005)). This algorithm is divided in two parts. The first one is similar to MDR but it uses visual information in order to be more accurate in finding data regions because it builds trees with visual information (left, right, top, bottom limit) by calling the embedded parsing and rendering engine of a browser. To build the tree they use the
sequence of opening tags and containment checks. The second part gives the possibility of aligning the data records when they are certain about the label, if not it doesn’t compromise. It is possible to use heuristics to choose an output that doesn’t contain all the data regions but just the ones specified.

**Identify data region**

Data region identification is similar to MDR, see section 3.1.1, but more restrict because it assumes that any gap in a data record should not be bigger than the gaps between data records.

**Data segmentation**

It is done in the same way as MDR, see section 3.1.1.

**Data alignment**

In the beginning, it is necessary to build a tree for each data record. It may be necessary to add an auxiliary root node if the data record is not present all together in the same sub-tree. Then, it uses the Partial Tree Alignment algorithm. At startup, a seed tree is built being the same as the tree with most data fields. For each node in each of the other trees, the algorithm tries to find a corresponding node in the seed tree. If it doesn’t exist, there are four possibilities.

First, if the nodes that don’t exist in the seed tree are encapsulated by two nodes that exist in the seed tree, then they are added to the seed tree between those nodes. As can be seen in Figure 3.7 (A).

Second, if the nodes that don’t exist in the tree seed are encapsulated in the left by a node that exists in the seed tree, then they are added to the seed tree on the right of that node. As can be seen in Figure 3.7 (B).

Third, it is similar to the second case but instead of being encapsulated in the left, the node is encapsulated in the right and then, it is added on the left of the existing node.
3.2. VISUAL-INFORMATION-BASED APPROACH

Beyond these three cases we can not decide where to insert the node because the insertion is ambiguous. That’s why it is a partial alignment. As can be seen in Figure 3.7 (C). To solve this limitation, the algorithm has more than one iteration that allows the seed tree to be more complete in the following iterations and to be able to consume nodes that had ambiguous insertions in the previous ones.

![Figure 3.7: Partial Alignment - Inserting nodes in the seed tree: (A) and (B) Unique expansion; (C) Insertion ambiguity. (Zhai & Liu (2005))](image)

3.2.2 Nested data Extraction using Tree matching and visual cues

Nested data Extraction using Tree matching and visual cues (NET) is an improvement of DEPTA (Liu & Zhai (2005)).

First it builds a tree using visual information to simplify the process. Then, it identifies and extracts data records. To build the tree it is necessary to identify the four boundaries of each HTML element and to find relationships between elements - containment relationships.

As MDR, see section 3.1.1, it assumes that:
1. A group of data records that contains descriptions of a list of similar objects are typically presented in a contiguous region of a page and are formatted using similar HTML tags.

2. A group of similar data records being placed in a region is reflected in the tag tree by the fact that they are under one parent node.

NET looks for repeating patterns starting in the lowest levels and going up successively after analyzing all the level. This guarantees that nested data records are found.

The result of each data region is put in different tables of a database.

**Identify data region and Data segmentation**

NET traverses the tree in post-order and applies Simple Tree Matching (Yang (1991)) to every pair of child nodes. This step makes it possible to find nested data records.

**Data alignment**

It is possible to perform data alignment after the tree matching, using the resultant matrices to find matched items in the two trees. If there is more than one node with a maximum of alignments, the first one is chosen.

### 3.2.3 Vision-based Data Extractor

Vision-based Data Extractor (ViDE), is based in the information that human beings see, i.e., location, size and font of the elements (Liu et al. (2010)). It also uses some information that is hidden from users like data types and frequent symbols. ViDE transforms the web page into a Visual Block tree, as we can see in Figure 3.8.

To process the web page, this system uses various features, namely (i) position, (ii) layout, (iii) appearance and (iv) content.

Position features include:

1. Data regions are always centered horizontally.
2. The size of the data region is usually large relative to the area size of the whole page.

Layout features include:

1. The data records are usually aligned flush left in the data region.
2. All data records are adjoining.
3. Adjoining data records do not overlap, and the space between any two adjoining records is the same.

Appearance features include:

1. Data records are very similar in their appearances, and the similarity includes the sizes of the images they contain and the fonts they use.
2. The data items of the same semantic in different data records have similar presentations with respect to position, size(image data item), and font (text data item).
3. The neighboring text data items of different semantics often (not always) use distinguishable fonts. This feature didn’t hold so well.

Content features include:

1. The first data item in each data record is always of a mandatory type.
2. The presentation of data items in data records follows a fixed order.
3. There are often some fixed static texts in data records, which are not from the underlying Web database. This feature didn’t hold so well.

**Identify data region**

The data region corresponds to a block and it is found using the position features and if more than one is selected, it chooses the one with the smallest area.
CHAPTER 3. RELATED WORK

Data segmentation

Each data record correspond to one or more subtrees in the Visual Block tree, which are child blocks of the data region. To find them, ViDE removes the top and bottom blocks of the data region which are not aligned to the left. Then, similar blocks are clustered. After, blocks must be regrouped to form data records considering that the number of data records corresponds to the number of blocks of the cluster with the first data items. After this, all the data records are identified. and each leaf node is a data item. Then if the data items have different fonts, it is possible to segment them into more data items.

Data alignment

It uses visual similarities (font and position) to align data items. Knowing that the order of data items in data records is fixed. If a data record does not have an optional data item, ViDE will fill the vacant position with a blank item, so, in the end, all data records have the same number of data items. An example can be seen in Figure 3.9. The different
3.2. VISUAL-INFORMATION-BASED APPROACH

shapes represent different data items.

At start (a) all data items are not aligned. However the first column is composed by data items of the same type, so they are clustered together. For the second column (b), there is a conflict of data items which form two clusters and to choose which one is in the correct position, the next columns must be checked. The cluster with the circles has a correspondence in column four, but there is a conflict because row one is already fulfilled with the same data item. The cluster with the diamond has a correspondence with column three and it doesn’t have a conflict because row two does not have a diamond. And so, it is decided that column two is formed by the circles and column three by the diamonds. The same is done for the following data items as we can see in (c) and (d).

![ViDE data items alignment example](Liu et al. (2010))

3.2.4 Visual information aNd Tag structure based wrapper generator

Visual information aNd Tag structure based wrapper generator (ViNTs) focuses on extracting information from search engine web pages (Zhao & Meng (2005)). For instance, www.google.com.

The wrapper must specify the minimal sub-tree and the separators. To choose the best wrapper, they use four heuristics based in visual information considers that the data region must:

1. Occupy a large area
2. Be centrally located
3. Contains many characters

4. Have large number of records

**Identify data region**

It divides the web page in sections containing lines and classifies these lines according to their content and calculates their position. The position is the left x coordinate. Blank lines are also counted because they give information about the visualization and their position is the same of the last line that was not blank. The position enable us to know the shape of a block. An example can be seen in Figure 3.10, where we can also verify that the first and fourth line are of the type LINK, the second and third line are of the type TEXT and the fifth line is blank, BLANK. With the classification and position of the lines composing the block it is possible to compare blocks.

![Financial Aid Forms](http://bingfa.binghamton.edu/navigation bar)

State residents) Click here to see the list of

http://bingfa.binghamton.edu/forms.htm

**Figure 3.10:** Block and correspondent shape example (Zhao & Meng (2005))

**Data segmentation**

The next problem is to find lines that form a data record separator. It can happen that the separator is multiple lines together. To do this, the system selects all the multiple lines patterns that appear at least three times. A suffix tree is used to find all the the possible separators in a simple manner. Using the separators found, the web page is segmented into several groups of blocks. Each of the groups are composed by similar blocks. The next step is to find the first line of data records in each block for the reason that if we find out the first line of a data record we can segment the block and get all the data records. To do this, the authors created 4 heuristics.

1. The line following an HR-LINE is a first line.
2. If there is only one line starting with a number in a block, this line is a first line.

3. If only one line in a block has the smallest position, this line is a first line.

4. If there is only one blank line in a block, the line following the blank line is the first line.

There is a minimal sub-tree that contains all the data records. To choose the best data records separator three conditions must be met.

1. Appears in every data record

2. Appears in each data record exactly once

3. Appears at the end of the data record

3.2.5 Visual Perception-based Extraction of Records

As a first step, Visual Perception-based Extraction of Records (ViPER) uses the Mozilla browser and it is open source layout engine Gecko to convert the HTML pages into valid XHTML because many web pages are not even according to the syntax of HTML (Simon & Lausen (2005)).

ViPER builds a tree in which each tag element contains it’s left-corner coordinates (x and y), width and height, the trimmed text between tags is replaced by a tag element <TEXT>and all the HTML style tags attributes are ignored. It also constructs a plain tag sequence structure in which each element has a link with the corresponding element of the tag tree and vice-versa. The tag sequence is used to find patterns.

Identify data region

To find candidate data regions, ViPER seeks for similar sub-trees using tree edit distance. It uses primitive tandem repeats in order to find the repeated structures. Tandem repeats allows to find all the common subsequences of two sequences, and because they are primitive, if there is the case of having found (DD) two times and D four times, D is selected. This idea is used in order to built a matrix that enables the system to find
candidate data regions.

**Data segmentation**

Data segmentation is done by comparing visual histograms of the repetitive patterns found before. The valleys in the histogram corresponds to white spaces that can be used to segment the data records, as it can be seen in the example of Figure 3.11. Then, ViPER calculates the weight of the candidate data regions from their area and the one that has better results is selected.

Figure 3.11: Visual histogram (Simon & Lausen (2005))

**Data alignment**

Having the data records identified, ViPER uses the tag sequences of each data record to search the maximal unique matches (MUMs) and maximal multiple exact matches (multiMEMs). MultiMEMs are maximal string sequences which are present in all data records and MUMs are MultiMEMs which are present in all data records exactly once. ViPER chooses a non-overlapping sequence of MUMs such that the summed weights of the elements is maximal. With this, all the data records become structured into sub-sequences and using multiMEMs, ViPER is able to align the data items within these sub-sequences.
3.2.6 Pair-wise Alignment-based Data Extraction

Pair-wise Alignment-based Data Extraction (PADE) is able to extract data records when the web page has at least two data records and allows the construction of ontologies in a fairly accurate way (Pku-Hkust et al. (2009)).

**Identify data region**

To be able to find the data region, the web page has to have several collections of similar data records. It searches the tag tree for similar children nodes under the same parent because it assumes that similar data records have the same parent node and consequently they form a data region. The similarity between nodes is made using normalized edit distance over their tag strings. There is a threshold to evaluate the similarity. The problem of evaluating the similarity in this way is two nodes that have the same structure but different number of repetitions in their subparts can be considered not similar. It uses tandem repeats to solve this problem. Then, it segments the data regions obtained before using the tandem repeats and the visual gap between segmented records. Compares the segmented records and merge data regions that have similar data records, although they may have different parent nodes. After this step, it’s possible to have several data regions and the system chooses the one that goes accordingly to three heuristics.

1. Data region is usually located at the center: a center distance is calculated between the center of the page and the center of the data region for each possible data region (Zhao & Meng (2005)).

2. Data region usually occupies a large space: the area of each data region is calculated (Zhao & Meng (2005)).

3. Data records usually contains more text that nodes in other sections

The chosen data region is the one that conforms better with these heuristics.
**Data segmentation**

It is done by finding repeated patterns and when more then one pattern is found, it chooses the one that conforms with the gap between data records is not smaller than the gap within data records.

**Data alignment**

Every two different records are aligned first because data values of the same attribute usually have the same data type and are similar to each other. Then it finds a global alignment that has the biggest number of pair-wise alignments. At last, it runs a method over nested data records.

**Data labeling**

PADE does it in the same way as DeLa, see section 3.1.3.

### 3.3 Ontology-Based Approach

Ontology-based systems have a different vision of how to approach the problem of information extraction from web pages, considering that having a manually or automatically constructed ontology is an essential step to solve it. Therefore, the first step is to build an ontology of the domain in question that will enable the rest of the process.

#### 3.3.1 Conceptual-Model-Based Data Extraction from Multiple-Record Web Pages

Conceptual-Model-Based Data Extraction from Multiple-Record Web Pages, or BYU-TOOL as most of other authors call this system, is able to extract data from unstructured documents, but they have to be rich in recognizable constants (for example: dates, names, ...), narrow in ontological breadth (ontology with a relatively small domain) and the concepts are about a main entity (Embley et al. (1999)). The system is not affected
by changes in the web page because it uses relationships between data items, cardinality constraints for these relationships, regular expressions that limit the possibilities for each data item and possible context keywords to help to find data items.

The algorithm goes as follows:

1. Manually build an ontology about the area of interest;
2. Using the ontology built before, create a database scheme and rules for matching constants and keywords;
3. Divide the document into several data records without the markup-language tags;
4. Extract data items from each data record using the matching rules;
5. Populate the database scheme.

Although the ontology is built manually, this system is considered unsupervised because the wrapper generation is automatic once the ontology has been written. If the domain changes, a new ontology has to be written. The rest remains the same.

**Identify data region**

It is made by using HTML tags. The HTML tree is built and then it searches for the subtree that is most likely to have data records. This is made by choosing the subtree with the highest fan-out, i.e. the subtree with the most child tags.

**Data segmentation**

BYU-TOOL searches for the most likely separator of records in the subtree selected before. The most frequent child tags are considered possible separators. The concept of frequent tag is not very explicit. The choice is made by combining five heuristics that allows to rank the different candidate tags.

1. Highest Count: ranks the candidates according to the number of tag occurrences.
2. Identifiable Separator: ranks the candidates according to a predetermined list of common HTML separator tags.
3. Standard Deviation: ranks the candidates according to the standard deviation of the text between identical tags. Because it assumes that records are similar in size.

4. Repeating pattern: ranks the candidates according to how often a particular tag appears comparing to a pair composed by the same tag and its first child tag.

5. Ontology Matching: ranks the candidates according to how close their number of occurrences corresponds to the average of the number of data items that should occur only once.

After the tag separator is found, a delimiter is inserted after each occurrence of the tag separator and all the HTML tags are removed. The next steps threat the record as an unstructured document.

Data alignment and labeling

It transforms the previous document in a intermediate table that have some (a lot) of conflicts and ambiguities using the ontology’s keywords and concepts. To solve those problems, the system uses three heuristics.

1. Singleton: uses keyword proximity for values that can occur at most once. The system rejects the record if it doesn’t find any possibility for a value that must be present.

2. Functional-group: assumes that functionally dependent objects occur in close proximity. Keywords belonging to other contexts define context switches.

3. Nested-group: is used to handle with n-ary relationships.

Problems are related to Natural Language processing and the difficulty of dealing with ambiguity in unstructured documents.
3.3.2 Ontology-assisted Data Extraction

Ontology-assisted Data Extraction (ODE) is able to build an ontology based on some assumptions (Pku-Hkust et al. (2009)):

1. Attribute labels are usually similar even for different web sites and there is a small number of possible different labels.
2. Each attribute usually has the same data type.
3. The sequence of data items within data records of different web pages usually follow a similar order.
4. Synonymous labels are found comparing the same attribute in different web pages.
5. Query interface and query results pages give a lot of useful information for the labeling process.

First it builds an ontology automatically and the rest of process goes from this first step.

Ontology construction

It is made by extracting data records from a collection of query result pages within the same domain and align them into a query result table. Then assigns a name to each column of the table using information from the query result pages and query interfaces. Next it matches the columns of the query result tables and query interfaces and using these matchings it is able to create an ontology for this specific domain.

First it uses the PADE wrapper, see section 3.2.6. Then, to label columns of the database table it uses four heuristics studied for DeLa, see section 3.1.3.

1. Match query interface element labels to data values.
2. Search for voluntary labels in table headers.
3. Search for voluntary labels encoded together with data values.
4. Label data values in conventional formats.
Next, matching techniques between columns of tables and query interfaces from different web pages are used.

Finally the ontology is created organizing the matchings identified.

**Identify data region**

It is performed by finding the subtree that has more correlation with the ontology. First it uses PADE since it is precise in finding data regions with two or more data records. The improvement comparing to PADE is that there are three possibilities after PADE returns and it chooses the best option according to which situation is in.

The possibilities are PADE returns a data region that is not the correct one, or does not return anything because the web page has just one data record or there is actually none. To solve this problem ODE follows the following steps. First, if there are other data regions that also has good correlation with the ontology and returns the data region. When there is no other data regions with good correlation with the ontology, searches the subtree with the biggest correlation with the ontology and returns the single data record (if any). If none of the previous steps give good results, then is assumes that there is no data records in the web page.

**Data segmentation**

It is just performed if there is more than one data record in the data region found in the last step. First, it finds C-repeated patterns. If just one C-repeated pattern is found, it assumes that each repeat is a data record. If more than one is found, then it chooses the one that better satisfies the heuristic defined by:

1. The visual gap between two data records should be no smaller than any gap between data items within a data record.

**Data alignment and labeling**

ODE does it by giving an attribute name of the ontology to each data item of a data record and then aligns data items with the same label.
3.4 Discussion

This section is a summary of the characteristics discussed in the previous sections. In Table 3.2, we present the systems ordered by year, referring the category and the minimum requirements needed to build the wrapper.

<table>
<thead>
<tr>
<th>System</th>
<th>Year</th>
<th>Category</th>
<th>Minimum Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYU Tool</td>
<td>1999</td>
<td>Ontology</td>
<td>At least 2 data records</td>
</tr>
<tr>
<td>RoadRunner</td>
<td>2001</td>
<td>Tag Structure</td>
<td>At least 2 web pages</td>
</tr>
<tr>
<td>MDR</td>
<td>2003</td>
<td>Tag Structure</td>
<td>At least 2 data records</td>
</tr>
<tr>
<td>DeLa</td>
<td>2003</td>
<td>Tag Structure</td>
<td>At least 2 web pages</td>
</tr>
<tr>
<td>Depta</td>
<td>2005</td>
<td>Visual Information</td>
<td>At least 2 data records</td>
</tr>
<tr>
<td>NET</td>
<td>2005</td>
<td>Visual Information</td>
<td>At least 2 data records</td>
</tr>
<tr>
<td>ViPER</td>
<td>2005</td>
<td>Visual Information</td>
<td>At least 2 data records</td>
</tr>
<tr>
<td>ViNTs</td>
<td>2005</td>
<td>Visual Information</td>
<td>At least 4 data records</td>
</tr>
<tr>
<td>PADE</td>
<td>2009</td>
<td>Visual Information</td>
<td>At least 2 data records</td>
</tr>
<tr>
<td>ODE</td>
<td>2009</td>
<td>Ontology</td>
<td>At least 2 data records</td>
</tr>
<tr>
<td>ViDE</td>
<td>2010</td>
<td>Visual Information</td>
<td>At least 2 data records</td>
</tr>
</tbody>
</table>

Most of the systems need at least two data records to build a wrapper because they do comparisons over the same web page to try to find a pattern. If there are not at least two data records, it is impossible to find any pattern. RoadRunner needs two web pages because it uses the dissimilarities found between the two to retrieve the information of interest and DeLa needs it, to be able to identify the data-rich section of the web pages. ViNTs needs four data records in order to segment the data records. It is also important to notice that ODE only needs at least two data records per query result page for building the ontology. After the ontology is built it doesn’t matter how many data records exist in the web page.

It is also shown the categories of each system: (i) text-based and tag-structure-based, in which we can find RoadRunner, MDR and DeLa, (ii) visual-information-based, where we can find Depta, NET, ViPER, ViNTs, PADE and ViDE and (iii) ontology-based represented by BYU Tool and ODE.

Table 3.3 shows the tasks that each system is able to perform. All the systems identify and segment the data region, however ViNTs and MDR don’t align the data items and
just a few systems are able to label the extracted information.

<table>
<thead>
<tr>
<th>System</th>
<th>Data region identification</th>
<th>Data segmentation</th>
<th>Data alignment</th>
<th>Data labeling</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYU Tool</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>RoadRunner</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>MDR</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>DeLa</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Depta</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>NET</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>VIPER</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ViNTs</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>PADE</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ODE</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ViDE</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### 3.5 Summary

Chapter 3 presents the state of the art in unsupervised web data extraction dividing the studied systems into three categories and explaining heuristics that were used to develop our features.
Chapter 4

GEN-X2

In this Chapter we present our system GEN-X2 which follows the Visual-Information-Based approach. It was designed to clean HTML web pages converting them to XHTML web pages, to create a grammar corresponding to the collection of web pages and to return a XPath expression composed by a various number of subexpressions which extracts information from the web page given as input using the grammar inferred to do so. That we are aware, this is the first time Genetic Programming and XPath are combined in order to extract information from web pages.

The following subsections present some important concepts to understand our approach, describe the system architecture and each component individually and define how the fitness function evaluate each XPath expression created.

4.1 Expressions, Subexpressions and Elements

In this project we approached the problem using Genetic Programming and XPath and our objective is to get a XPath expression which is able to retrieve information from the web page trying to ignore redundant information. For this, the XPath expression must be complete, i.e., a single XPath expression has the ability to retrieve all the information of the web page. This means the XPath expression is composed by a variable number of subexpressions as we can see in figure 4.12.
Each subexpression must retrieve one data attribute and only one.

The final XPath expression returned must be a combination of subexpressions that should be better than any isolated number of subexpressions or different incomplete combination of them. The order of the subexpressions does not matter for the result.

Elements correspond to HTML elements and they may have text or not and have an area (it can be 0 when it is not a visible part of the web page). A XPath expression/subexpression can return zero, one or multiple elements.

XPath expressions with redundant subexpressions return the same number of elements. In figure 4.13 we can see an example of subexpressions redundancy.

The subexpressions are expected to extract tokens, i.e., the text that is within the elements.

## 4.2 System Architecture

The system architecture is presented in figure 4.14 and has the following steps. First, we convert the HTML web page possibly with some errors (that browsers don’t care about but the parsers do) to XHTML. Examples of transformations are described in Section 4.2.1. After we get the clean and valid XHTML we render the web page using PhantomJS\(^1\) and insert the coordinates of each HTML element (left, top, right, bottom, \footnote{\texttt{http://phantomjs.org/}}
width and height) in the web page. Besides width being simply right minus left and height
being bottom minus top, we chose to put this "redundant" information to simplify the eval-
uation of XPath expressions in the next steps. After this, we create a BNF grammar of
our collection of input web pages already converted and rendered (XHTML web page
with position of every element). Then, we use GEVA as a mean to solve our problem
of finding which XPath expression is better suited to retrieve good information, ignoring
redundant information.

![Figure 4.14: System architecture](image)

### 4.2.1 Clean HTML Web page

We chose to clean the HTML web page given as input because XML DOM Parser can
give errors while processing the page even if browsers don't complain about it. For this,
we use JTidy \(^1\) to check and clean the HTML web pages. JTidy is able to fix some
common errors like the ones listed above.

- Convert to lowercase
- Correct elements nesting

\(^1\)http://jtidy.sourceforge.net/
• Correct attributes

• Close all opening tags

• Encode “<“ and ”&” symbols

For performing this task we also tried to use jsoup \(^1\) but it showed to change the given web page too much transforming the input to an almost completely different web page. For reasons of trying to maintain the web page as similar as possible to the given input but with no error we choose JTidy as a better tool for doing this in our context.

4.2.2 Web page Renderer

To use visual information to evaluate the XPath expressions, first of all we have to render the web page in question and get the positions of every element returned by the expression. For doing this, we use PhantomJS, which is a headless browser based on Webkit scriptable with a JavaScript API. This means PhantomJS works like a browser but it does not have a graphical user interface. Since, we just want to get the position of every element of the webpage this is perfect. We obtain all elements position using JavaScript and write the information as an attribute of the element. This way when parsing the XHTML web page we can get the position of the element in question right away without having to render the web page each time we need this information.

We also tried to use JRex \(^2\) for this purpose. It gives the possibility of embedding Mozilla Browser within a Java Application. However, JRex revealed to be a solution with a lot of problems when the web page was more complex and with more incorporated scripts. For this reason we chose PhantomJS has a mean to achieve this goal. Also, Google Chrome and Safari are browsers based on Webkit and since Google Chrome is the browser most used by Internet users, according to StatCounter \(^3\), this seems to the best tool with the results nearer to what most Internet users see when browsing.

\(^1\)http://jsoup.org/
\(^2\)http://jrex.mozdev.org/docs.html
\(^3\)http://gs.statcounter.com/#browser-ww-yearly-2008-2013
4.2. SYSTEM ARCHITECTURE

4.2.3 Grammar Creator

A BNF grammar must be created so that GEVA\(^1\) can create individuals (XPath expressions), evaluate them and transform them using operations of mutation and crossover.

We created a grammar by creating all the unique XPath expressions and evaluating them and organizing them from best to worst. The unique XPath expressions are created by finding all elements containing text and adding to the XPath expression the nodes of the parents and the class and id attributes if they are present. If you get to the root node, and the expression is still not unique, then the position of the element is added, starting from the end to the beginning. We consider an expression unique, when it just returns one node.

In addition to the unique expressions, the expression created before adding positions (indexes) is also added. In this way we intend to be able to catch data items with multiple values.

If the web page has tables with only two columns, more expressions are also added. The expression will be the expression which retrieves the information of the second column considering that the first column contains the present value. There will be created as many expressions as the rows of the table. In this way we intend to extract correctly data items in which the position (index) is not enough for the correct extraction of the item for the reason that table does not always have the same size through all the pages of the collection.

After having all the unique XPath expressions, we evaluate them using the fitness function created and organize them from best to worse in the following manner.

The grammar is composed by expressions with a variable number of subexpressions. In our case, we limited to 20 subexpressions because the more subexpressions an expression has, the more time is needed for processing and 20 seemed a good limit since we almost always want to try to catch only 4 or 5 data items from a web page.

Then, the expression is composed by several groups of five expressions in which, the next group is composed of all previous groups and five new expressions.

\(^1\)http://ncra.ucd.ie/Site/GEVA.html
We tried different grammars to see which one worked better and figured out that the one described above was the best for our purpose, not only in terms of obtained results but also in terms of performance.

### 4.2.4 GEVA

GEVA (O’Neill *et al.* (2008)) is a Grammatical Evolution in Java tool developed at UCD’s Natural Computing Research & Applications group, which enable us to evolve a program written as a BNF grammar and a fitness function. We use it to create and evolve XPath expressions.

With GEVA we can control many parameters, such as the population number, the way we initialize it, the fitness function and the mutation and crossover functions. However, one parameter that couldn’t be change was the way GEVA selected the fittest (best) individual. GEVA always choose the individual with the minimum fitness value and zero (0) is supposed to be the best result possible. This was a limitation to our approach. In the beginning we try to use it using the fitness function tending to a minimum but after a lot of thought we changed GEVA to support maximum fitness too, passed as a parameter. In this system it makes more sense to evaluate the expressions considering a gain of information than going to a minimum with all the subsequent problems of normalization.

### 4.2.5 Labeler

From the expression obtained from GEVA, we save the result of each subexpression in a database and try to label it. To do that, first, we check if the subexpression is part of a table (ends with td tag). If that is the case, we try to find if there is a possible label inside it, i.e., if there is only one tag inside it and it has the same value over all the web pages of the collection then that value is a possible label. Other possibility is, if the table is composed by two columns and the first column has the same value over all the web pages of the collection, then that value is a possible label for the the values found in the next column. And finally we check if there is a common prefix for all the obtained tokens. There is no possible label if none of the cases described above is met.
4.3 Fitness Function

In this Section we describe how the fitness function is defined. This function gives a rating to each XPath expression that is generated by the program through initialization, mutation and crossover operations. It has the goal of refining the XPath expressions during the course of the run. As said before, the fitness function is maximized, so 1 is better than 0.

\[ \text{FitnessFunction} = \sum_{i=1}^{12} W_i \times F_i \]  

(4.3.1)

\( W_i \) can only have two values, 0 or 1, representing if the specific feature will be used or not and \( F_i \) is the value obtained using a specific feature. The calculation of each feature is presented in the following subsections.

4.3.1 Feature 1 - Centered information is better

This feature represents the idea that useful information usually is near the center of the web page and to calculate it we need to get the average element distance to the center and the maximum distance to the center. The maximum distance to center is easy to find because we just need to know the middle point of the web page to get it. To get the average element distance, we need to get the top-left corner and the bottom-right corner of each element returned by the expression.

For each subexpression and for each element we get the average distance to the center by averaging the distance to the center of the top-left corner and bottom-right corner. Then, we calculate the average distance of all elements to the center of each subexpression. The value obtained will be between 0 and 1 and we subtract this value to 1, so the 1 means a good individual (centered one) and 0 means a bad one.

This feature will have a value between 0 and the number of subexpressions which compose the expression being evaluated.
Feature 1 = \[ \sum_{i=1}^{\text{subexpressions number}} \left( 1 - \frac{\text{Average element distance}}{\text{Max distance}} \right)_i \] (4.3.2)

The distance between two points is calculated through the following equation:

\[ \text{Distance between 2 points} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \] (4.3.3)

4.3.2 Feature 2 - Big area is better

This feature represents the idea that useful information usually occupies a great part of the web page.

\[ \text{Feature 2} = \sum_{i=1}^{\text{subexpressions number}} \left( \frac{\text{Element Area}}{\text{Page Area}} \right)_i \] (4.3.4)

We faced some problems when calculating the element area. First, we used a rough approach that simply looked for the biggest rectangle and calculated the area based on it. However this is not a good approach because it gives a poor approximation. For solving this, we used a line sweep algorithm as stated by Shamos & Hoey (1976). Figure 4.15 shows the algorithm in action.

First of all the rectangles must be aligned by x-axis and then by y-axis. It is possible to calculate the union of the rectangles using events and an active set. For each rectangle there are two events corresponding to the start and end of the rectangle (left and right edge). When we get a start point, the rectangle is added to the active set and when find the end point the rectangle is removed from the active set. The active set represents the rectangles that influence the area in this window. With this information we can get the length of the segment that is activated. Then we just need to multiply the segment length by the distance between events. As we can see in figure 4.15.

This feature will have a value between 0 and the number of subexpressions which compose the expression being evaluated.
Figure 4.15: Line sweep algorithm example
4.3.3 Feature 3 - More characters is better

This feature intends to catch the information present in the web page that is represented by “visible” characters. With this in mind, the more characters we can get, the better.

\[
Feature_3 = \sum_{i=1}^{\text{number of subexpressions}} \left( \frac{\text{Element Characters}}{\text{Total Characters}} \right)_i \tag{4.3.5}
\]

The result will be a value between 0 and the number of subexpressions which compose the expression being evaluated.

4.3.4 Feature 4 - No area intersection is better

This feature penalizes expressions composed by subexpressions whose area overlap. Although the bigger the area the better, we don’t want overlapped area and this feature does it by summing the area of each element and divide by the union area of all elements (as it was calculated in Feature 2 presented in Section 4.3.2).

\[
Feature_4 = - \frac{\sum_{i=1}^{n} \text{Element Area}_i - \text{Element Area}}{\text{Element Area}} \tag{4.3.6}
\]

The objective of this feature is only penalizing and having no effect in case there is no overlapping.

4.3.5 Feature 5 - Subexpressions must have only one type

This feature verifies if the subexpressions only have one type penalizing features with no type (whitespaces, no text). Several types are identified: e-mail, datetime, monetary values, measurements, numbers, phone or fax, websites. We are also able to identify people, organizations, locations and sports teams using Freebase API (Section 2.4.1) by comparing if the token is equal to the name or alias of the entities found by making a search using the token as keyword. When it is not possible to identify any type from the token, an unknown type is assigned.
4.3. **FITNESS FUNCTION**

4.3.1 Feature 5 - Subexpressions number

$$ Feature_5 = \sum_{i=1}^{\text{subexpressions number}} \left( \text{Number of Types}_i \right) \quad (4.3.7) $$

where

$$ \text{Number of Types}_i = \begin{cases} 
1, & \text{if same type} \\
0, & \text{if same type > unknown} \\
0, & \text{if all unknown} \\
-1, & \text{if otherwise} 
\end{cases} \quad (4.3.8) $$

4.3.6 Feature 6 - Less subexpressions is better

This feature adjust the expressions to be more compact, i.e., less subexpressions are better than a lot of them that retrieve the same amount of information. As we do not know the ideal number of subexpressions, we use a threshold of 5% of all subexpressions present in the grammar, as an acceptable number of subexpressions.

$$ Feature_6 = \begin{cases} 
\text{Acceptable Subexpressions}, & \text{if Subexpressions Ratio} \leq 0.05 \\
\text{Penalization}, & \text{otherwise} 
\end{cases} \quad (4.3.9) $$

where

$$ \text{Subexpressions Ratio} = \frac{\text{Number of Subexpressions}}{\text{Total Number of Subexpressions}} \quad (4.3.10) $$

$$ \text{Acceptable Subexpressions} = \text{Total Subexpressions Number} \times 0.05 \quad (4.3.11) $$

$$ \text{Penalization} = \text{Acceptable Subexpressions} \times (1 - \text{Subexpressions Ratio}) \quad (4.3.12) $$
In case the number of subexpressions is less than the number of acceptable subexpressions, this feature has the value of the acceptable subexpressions. With this we don't penalize expressions with a number of subexpressions minor to the threshold. Otherwise, the feature has a value of the acceptable subexpressions number minus the subexpressions ratio times the threshold. This way, the more subexpressions, the worst the expressions is.

4.3.7 Feature 7 - Deep subexpressions is better

Usually the information is in the leafs of the web page and this feature goes in that direction. Deeper subexpressions are better than superficial ones. With this we intend to get more accurate subexpressions.

\[
Feature_7 = \frac{\text{Subexpressions Depth Average}}{\text{Grammar Maximum Depth}} \times \text{Subexpressions Number} \quad (4.3.13)
\]

4.3.8 Feature 8 - Expression possible in more web pages is better

With this feature we try to select global expressions, i.e., expressions that retrieve text in more web pages of the collection is better than an expression that retrieves text only in one web page of the collection.

\[
Feature_8 = \text{Expression Possible Number of Web pages} \quad (4.3.14)
\]

This feature is especially important when the grammar has many expressions that don’t extract anything.

4.3.9 Feature 9 - Distinct tokens are better

This feature intends to give better values to subexpressions which return different tokens, i.e., tokens that are different in all the web pages given as input. The purpose is to eliminate menus and labels and get only different text.
4.3. FITNESS FUNCTION

Feature 9 = \[ \sum_{i=1}^{\text{subexpressions number}} (\text{Different Tokens}_i) \] (4.3.15)

where

\[ \text{Different Tokens}_i = \begin{cases} -1, & \text{if all the same} \\ 0, & \text{if only 1 token} \\ 1, & \text{otherwise} \end{cases} \] (4.3.16)

4.3.10 Feature 10 - More subexpressions possible is better

Similar to feature 8, this feature tries to find expressions composed by subexpressions that return text in more web pages of the collection.

\[ \text{Feature 10} = \text{Possible Number of Subexpressions} \] (4.3.17)

4.3.11 Feature 11 - Subexpressions must be different

Similar to feature 4, this feature intends to eliminate subexpressions with overlapping values, i.e., if there is overlapping values we want to be able to select the best one and not the two of them.

Feature 11 = \[ \sum_{i=1}^{\text{subexpressions number}} (\text{Repetitive subexpression}_i) \] (4.3.18)

where

\[ \text{Repetitive subexpression}_i = \begin{cases} -1, & \text{if repetitive} \\ 0, & \text{otherwise} \end{cases} \] (4.3.19)

In addition, if the subexpression is repeated, the values of all the features of that subexpression are also subtracted. The subexpression is considered repetitive if there is an overlap of the extracted tokens.
4.3.12 Feature 12 - Subexpressions have a possible label

This feature tries to find a possible label to the tokens extracted by each subexpression.

\[
Feature_{12} = \sum_{i=1}^{\text{subexpressions number}} \text{Possible label}_i
\]  

(4.3.20)

where

\[
\text{Possible label}_i = \begin{cases} 
1, & \text{if possible label} \\
0, & \text{otherwise}
\end{cases}
\]  

(4.3.21)

The possible label is found using the same approach as described in Section 4.2.5.

4.4 Summary

Chapter 4 presents GEN-X2, the system created throughout this dissertation, explaining every module in detail and giving special attention to the fitness function, the function which is able to evaluate and rank XPath expressions.
Chapter 5

Validation and Experimental Results

Methods used to validate the proposed system and the chosen dataset are described in Chapter 5. The experiments taken to assess the performance of GEN-X2 are also described.

5.1 Validation

The measures used to evaluate the system were Precision and Recall.

- **Precision** has the objective of evaluating the ratio of correct answers (CA) over the total answers (TA).

\[
    \text{Precision} = \frac{CA}{TA} \quad (5.1.1)
\]

- **Recall** has the objective of evaluating the ratio of correct answers (CA) over the total answers that existed in the web page (TR).

\[
    \text{Recall} = \frac{CA}{TR} \quad (5.1.2)
\]
In the context of this thesis and considering that the result is an XPath expression with several XPath subexpressions, we consider correct answer if the subexpression which retrieves the correct information is present.

### 5.1.1 Datasets used in the Experiments

We could not find completely appropriate datasets, since our system wants to recover all the possible information inside a web page that is not a menu or ads. All available public datasets we found have specific data records and data items which should be extracted correctly and not all the possible information of the web page. For that reason we selected a dataset written in the format of GroundTruth\(^1\) with: a total number of 13,509 web pages were used from 5 different categories.

### 5.2 Results and Discussion

#### 5.2.1 Performance

To calculate precision and recall, several parameters were chosen after analyzing several runs. The parameters used were 10 input web pages, population of 400, 200 generations and the expression must not be composed by more than 20 subexpressions. Feature 5, 9, 11 and 12 were the features chosen in order to get better results. The input pages are selected randomly from a collection of pages with the same template. Due to the probabilistic nature of the system, the results are the average of 5 runs.

From the results obtained presented in Figure 5.16, we can verify that there is an average precision of 74.66% and an average recall of 76.06%. The low levels of precision occur when the subexpression selected was containing many more data items together. For example: when all the data is presented in a table and all the data items are in a td tag. If the subexpression obtained wasn’t specific enough, many more data items are catch together and for that reason the precision can be less than 50%. Because when that happens usually the table is quite big and has at least 8 data items. However we

---

\(^1\) [http://swde.codeplex.com/](http://swde.codeplex.com/)
preferred to leave this kind of expression in our grammar because it is the only way of catching multiple data items.

On the other hand, recall can be low because we limited the number of subexpressions to 20 and the population and generations number is not that high. Because of this, some subexpressions can never appear, or even if they appear during the running of the program, it can get repetitive subexpressions or the combinations with other subexpressions were not that good. So, it will not appear in the final result. And there is one more reason, from the dataset, we obtain certain data items, but the web page contains many more that may also be interesting. For example, in the dataset Fanhouse, we extract the players date of birth almost every time, which is a valid attribute and interesting in
many occasions. However, the dataset does not present this as a data item of interest to be extracted. As such, from the 20 subexpressions to withdraw good attributes, this subexpression takes a good attribute but does not contribute anything to the system’s recall.

GEN-X2 works quite well for simple pages and simple data items. For multiple data items it is a more difficult task that does not work every time.

Figure 5.17: GEN-X2 general performance
5.2. RESULTS AND DISCUSSION

5.2.2 General Performance

As said before, GEN-X2 is able to extract much more accurate information than the one presented in the dataset. For this reason, we did a more subjective analysis in order to realize to which extent the system was being able to extract correct data items and ignoring non-useful information.

This analysis consisted in verifying if the data extracted by the subexpressions was consistent. The data extracted was then classified into one of three categories (i) correct information (data item well extracted), (ii) mixed information (several data items of different types extracted together but that have the possibility of being correctly separated) and (iii) bad information (menus, ads or mixed information with no possibility to be separated correctly). The parameters used were the same described in Section 5.2.1.

From the obtained results presented in Figure 5.17, we verify that GEN-X2 is able to extract correct information 66.72% of the time on average and bad information 15.35%. Bad information can be captured when the menus/ads are different from page to page and might also be from a specific type. The biggest problem is with pages where the position is not enough to extract the correct data.

The mixed information occurs because most of the information can’t be attributed a specific type and as unknown information, everything is extracted together.

Figure 5.18: Extraction example
An example of an extraction is presented in Figure 5.18 and it is possible to verify that GEN-X2 is able to extract data items correctly and labeling them when a possible label is found.

5.3 Summary

In this Chapter we specify how the system was evaluated by calculating its precision and recall and stating which parameters were used for running it. There were two ways to evaluate the performance of the system and both of them are also described.
Chapter 6

Conclusions and Future Work

In this chapter we present the main contributions in Section 6.1, the conclusions in Section 6.2 and finally we point some possible directions that could be used for future work in Section 6.3.

6.1 Contributions

The main contributions of this work can be summarized as follows:

**GEN-X2**: the system developed during this thesis implements a new approach to the problem of web data extraction, i.e., it unifies Genetic Programming and XPath expressions as a mean to retrieve information in an automatic manner from general web pages. HTML web pages are cleaned in order to be parsed by Java DOM parser correcting some possible existing errors. The web page is rendered and the position of each HTML element is added as attributes of the element. A BNF grammar is created by the collection of the web pages given as input so XPath expressions can be generated through the GEVA (the grammatical evolution tool). After that GEVA returns the best XPATH expression found during the run. The system is available in my Bitbucket account.

---

1https://bitbucket.org/HugoGuimaraes


Set of Features to evaluate XPath expressions: ranks the XPath expressions according to the likelihood of containing relevant information. It can work as a ranking system, since it is able to characterize XPath expressions using visual (position, area), textual (quantity of text and repetitions) and self-contained (number of subexpressions and depth) features, which can be combined in endless ways.

Utils Library: that attempts to simplify the creation of BNF grammars from XHTML web pages (single or collection) and the acquisition of each HTML element position in the web page through the renderization of it. These were the most laborious tasks with fewer information and examples online, so we think it can be a great help to future users of this approach and it can be useful for other areas besides web data extraction as well. It is also available in my Bitbucket account.

6.2 Conclusions

The construction of the system GEN-X2 raised many problems which needed to be solved before the system could run. In addition to the problems inherent to web data extraction, the processing of web pages with errors and obtaining the coordinates of the elements of the web page are not that easy to solve because even the best publicly available libraries do not solve them completely.

Moreover, theoretically the use of Genetic Programming as a mean to generate XPath expressions seemed a good idea, since we do not know the expressions that we want to find. However, if we use a grammar too general (XPath BNF grammar) a tremendous amount of expressions that are not even present in one of the web pages of the collection will be created and evaluated requiring an enormous computational power and a big amount of generations and/or population in order to obtain reasonable results. Thus, we decided to process some web pages of the collection given as input in order to find possible XPath expressions greatly limiting the search space of expressions.

From the results presented in Chapter 5 showed that GEN-X2 can perform quite well considering it just have to get some webpages as input and it does all the work after that.

\footnote{http://www.w3.org/2002/11/xqueryxpath-applets/xpath-bnf.html}
6.3 Future Work

For future work, we think it would be interesting to study how the various features developed in the context of this system behave in a more limited environment, i.e., without using Genetic Programming. And new features could be added, for example, a feature that reflects that data items are normally close to other data items.

It would also be interesting to study the effect of changing some parameters, like number of input web pages, size of the population and number of generations in a more accurate way.

In addition to correcting the problems related to the evaluation of XPath expressions, two more things can be made in order to improve the performance of GEN-X2.

VTD-XML \(^{1}\) claims to be the fastest XPath 1.0 implementation and in fact by our experience it seemed much faster than the DOM parser, but it did not return the correct result in a simple function like: how to get the text of an element. If the element had an element and then the text, VTD-XML parser would fail in search of that text. Therefore it would be necessary to change this function to give the desired results that would be correct (at least in our case). Since some time is needed in order to fully understand how VTD-XML works, a library with some crucial functions can be created and compared with the DOM parser to check if it is a good possibility even for other projects that work over XML and XPath.

Similar to a parser update, the HTML cleaner and even the renderer can be updated if new ones give better results. At the time of the development of GEN-X2, Jtidy and PhantomJS seemed to be the best option for these tasks.

\(^{1}\)http://vtd-xml.sourceforge.net/
Bibliography


Information and knowledge management, CIKM ’05, 381–388, ACM, New York, NY, USA.


Appendix A

GEVA Parameters

Table A.4: GEVA input parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>For help</td>
</tr>
<tr>
<td>-v</td>
<td>For GEVA version</td>
</tr>
<tr>
<td>-mutation_probability</td>
<td>Change mutation probability</td>
</tr>
<tr>
<td>-crossover_probability</td>
<td>Change crossover probability</td>
</tr>
<tr>
<td>-initialiser</td>
<td>Change intialiser operation</td>
</tr>
<tr>
<td>-generations</td>
<td>Change number of generations</td>
</tr>
<tr>
<td>-replacement_type</td>
<td>Change the replacement type</td>
</tr>
<tr>
<td>-main_class</td>
<td>Change the main class to execute</td>
</tr>
<tr>
<td>-crossover_operation</td>
<td>Change crossover operation</td>
</tr>
<tr>
<td>-evaluate_elites</td>
<td>Choose if elites should be evaluated</td>
</tr>
<tr>
<td>-derivation_tree</td>
<td>Choose the type of derivation tree</td>
</tr>
<tr>
<td>-grammar_file</td>
<td>Choose grammar file</td>
</tr>
<tr>
<td>-grow_probability</td>
<td>Choose grow probability</td>
</tr>
<tr>
<td>-population_size</td>
<td>Choose population size</td>
</tr>
<tr>
<td>-output</td>
<td>Choose file to output</td>
</tr>
<tr>
<td>-fitness_function</td>
<td>Choose fitness function</td>
</tr>
<tr>
<td>-selection_operation</td>
<td>Choose selection operation</td>
</tr>
<tr>
<td>-mutation_operation</td>
<td>Choose mutation operation</td>
</tr>
<tr>
<td>-max_depth</td>
<td>Choose max depth of individuals</td>
</tr>
<tr>
<td>-elite_size</td>
<td>Choose elite size</td>
</tr>
<tr>
<td>-tournament_size</td>
<td>Choose tournament size</td>
</tr>
<tr>
<td>-initial_chromosome_size</td>
<td>Choose initial chromosome size</td>
</tr>
</tbody>
</table>
Appendix B

GroudTruth Dataset

The list of attributes which were supposed to be retrieved using GroundTruth dataset.
### Table B.5: General Performance in GEN-X2

<table>
<thead>
<tr>
<th>Category</th>
<th>Website</th>
<th>Attribute</th>
<th>Existing attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto</td>
<td>carquotes (2000)</td>
<td>engine 2000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>fuel economy</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>model 2000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>price 2000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>date posted 2000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>location 2000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>title 2000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>date posted 2000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>location 2000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>title 2000</td>
<td></td>
</tr>
<tr>
<td>NBA Player</td>
<td>fanhouse (446)</td>
<td>height 446</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>name 446</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>team 446</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>weight 446</td>
<td></td>
</tr>
<tr>
<td>Restaurant</td>
<td>usdinners (2000)</td>
<td>address 4322</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>cuisine 1999</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>name 2000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>phone 2000</td>
<td></td>
</tr>
<tr>
<td>University</td>
<td>collegeboard (2000)</td>
<td>name 2000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>phone 2000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>type 2000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>website 2000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ecampustours (1063)</td>
<td>name 1063</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>phone 1063</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>type 1063</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>website 1063</td>
<td></td>
</tr>
<tr>
<td></td>
<td>embark (2000)</td>
<td>name 2000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>phone 2000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>type 2000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>website 2000</td>
<td></td>
</tr>
</tbody>
</table>