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

Hugo Miguel Laranjeira Guimarães
hugo.guimaraes@ist.utl.pt
Instituto Superior Técnico, Taguspark
Lisboa, Portugal

October 2013

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 report 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

1. Introduction

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* [6].

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 which enables a user to query a hidden database and get useful information.

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.

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.

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 [23], is used to evolve a group of XPath expressions and return the fittest one.

2. Concepts

2.1. Web Data Structure

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 [8]. 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 : 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.

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 [18].

2.2. 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 HyperText Markup Language [3] (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.

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 [4] (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.

Because all of these languages are a W3C Recommendation [5], we believe these languages have more probability of continue being the most used online.

2.3. Web Data Extraction

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 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 [12], Minerva [7] and WebOQL [2].

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 [11], RAPIER [10], WIEN [16], WHISK [28], NoDoSE [1], SoftMealy [14], STALKER [21] and DEByE [17] [26].

Semisupervised WDE Systems : The system learns how to extract the information from non labeled examples given by the user. The

[1] http://www.w3schools.com/xpath/xpath_intro.asp
[2] http://www.w3schools.com/xml/xml_whatis.asp
[3] http://www.w3schools.com/html/html_intro.asp
[4] http://www.w3schools.com/html/html_xhtml.asp
[5] http://www.w3.org/

user has the post effort of choosing the target pattern and indicate the data to be extracted. For example: IEPAD [5], OLERA [4] and Thresher [13].

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.

2.4. Genetic Programming

Genetic Programming (GP) is a methodology that allow us to create a computer program from existing ones [25]. 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 1, 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 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

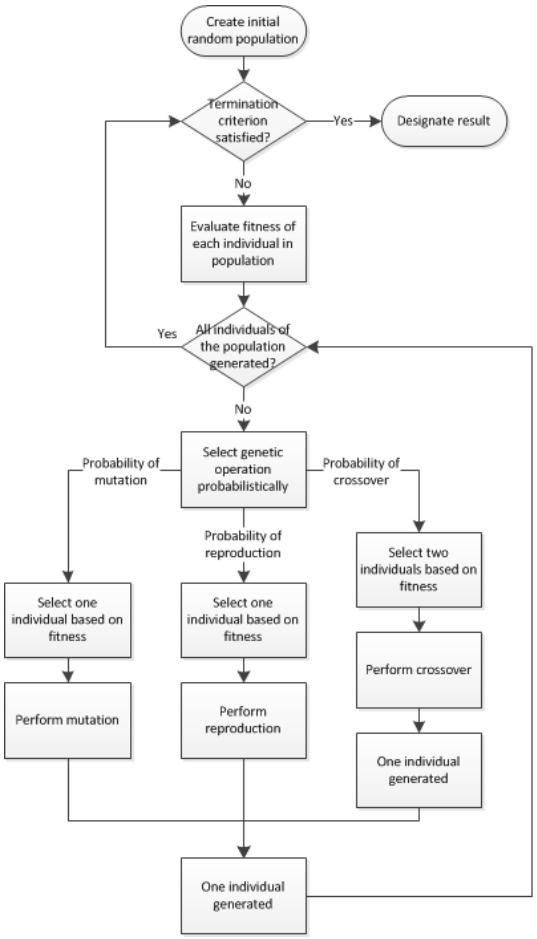

Figure 1: Genetic Programming overview. Adapted from [15].

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.

2.5. 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. The grammar is usually written in Backus-Naur form (BNF grammar).

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). 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.

2.6. 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 or 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 [6] and DBpedia [7]. 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.

Freebase was created with the intention of becoming a public repository of human knowledge scalable and is composed of diverse and heterogeneous information [3]. 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 a perpetually available service and it is accessible through an API with a big rate limit (100,000 per day).

3. Related Work

Several systems were studied and divided into three categories (i) Text-Based and Tag-Structure-Based Approach, where we can find MDR [18], RoadRunner [8] and DeLa [29], (ii) Visual-Information-Based Approach in which we have DEPTA [30], NET [19], ViDE [20], ViNTs [31], ViPER [27] and PADE [24] and (iii) Ontology-Based Approach where we can find BYU-TOOL [9] and ODE [24].

For the creation of GEN-X2, ViDE and ViNTs were specially important because of the heuristics they use to extract data items, like adjoining data records do not overlap, and the space between any two adjoining records is the same and data regions usually occupy a large area, is centrally located and contains many characters.

4. GEN-X2

4.1. Expressions, Subexpressions and Elements

In order to obtain a XPath expression which is able to retrieve information from the web page trying to ignore redundant information, the expression must be complete, i.e., a single XPath expression has the

[6] http://www.freebase.com/
[7] http://dbpedia.org/About

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 2.

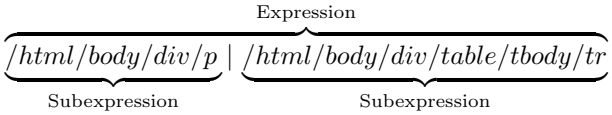

Figure 2: XPath expression and subexpressions example.

Each subexpression must be self-sufficient, i.e., must comprise 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 3 we can see an example of subexpressions redundancy.

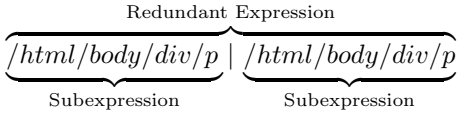

Figure 3: Redundant subexpressions return the same number of elements, like if it was just one present.

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 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. After we get the clean and valid XHTML we render the web page using PhantomJS [8] and insert the coordinates of each HTML element (left, top, right, bottom, 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 evaluation 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.

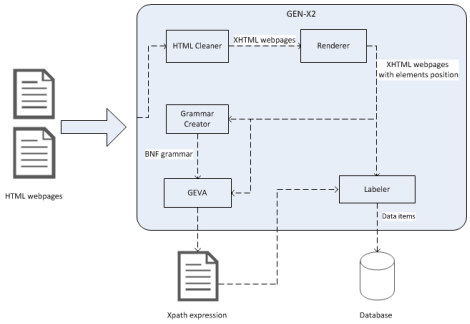

Figure 4: System architecture.

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 [9] to check and clean the HTML web pages. JTidy is able to fix some common errors like (i) convert to lowercase, (ii) correct elements nesting, (iii) correct attributes, (iv) close all opening tags and (v) encode "<" and "&" symbols.

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 web page 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.

[8] http://phantomjs.org/

[9] http://jtidy.sourceforge.net/

4.2.3 Grammar Creator

A BNF grammar must be created so that GEVA [10] 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 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.

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.

[10] http://ncra.ucd.ie/Site/GEVA.html

4.2.4 GEVA

GEVA [22] is a Grammatical Evolution in Java tool developed at UCDs 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 have control over 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

The fitness 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.

$$FitnessFunction = \sum_{i=1}^{12} W_i \times F_i \qquad (4.1)$$

Wi can only have two values, 0 or 1, representing if the specific feature will be used or not and Fi 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.

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.

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 this. However this is not a good approach because it gives a really bad approximation. For solving this, we used a line sweep algorithm. 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.

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.

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).

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 by comparing if the token is equal to the name or alias of the entities found by making a search using the token as keywords. When it is not possible to identify any type from the token, an unknown type is assigned.

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.

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 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.

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 are better than an expression that retrieves text only in one web page of the collection.

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.

4.3.10 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.

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.

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. It tries to do that as it is explained in Section 4.2.5.

5. Evaluation

To evaluate GEN-X2 performance we use precision and recall.

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

$$Precision = \frac{CA}{TA} \tag{5.1}$$

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

$$Recall = \frac{CA}{TR} \tag{5.2}$$

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 [11]. A total number of 13,509 web pages were used from 5 different categories.

5.1. Performance

To calculate precision and recall, several parameters must be chosen. 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.

From the results obtained, 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 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

[11] http://swde.codeplex.com/

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.

5.2. General Performance

As said before, GEN-X2 is able to extract much more accurate information that 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 5.1.

From the obtained results, 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.

An example of an extraction is presented in Figure 5 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.

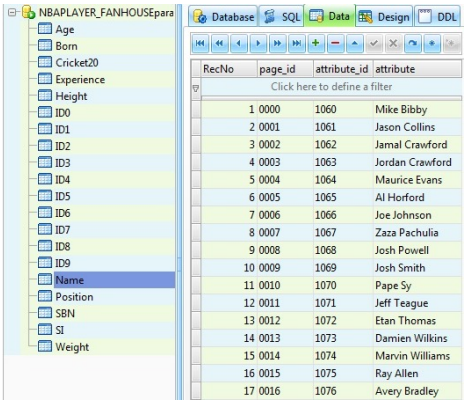

Figure 5: Extraction example.

6. 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) [12] 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.

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

References

[1] B. Adelberg. Nodose - a tool for semi-automatically extracting structured and semistructured data from text documents. In *SIGMOD Record*, pages 283–294, 1998.

[2] G. O. Arocena and A. O. Mendelzon. Weboql: Restructuring documents, databases and webs, 1998.

[3] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 1247–1250, New York, NY, USA, 2008. ACM.

[4] C.-h. Chang and S.-c. Kuo. Olera: A semi-supervised approach for web data extraction with visual support. *IEEE Intelligent Systems (SCI, EI*, 19:2004, 2004.

[5] C.-h. Chang and S.-C. Lui. Iepad: Information extraction based on pattern discovery. pages 681–688, 2001.

[12] http://www.w3.org/2002/11/xquery-xpath-applets/xpath-bnf.html

[6] K. C.-c. Chang, B. He, C. Li, M. Patel, and Z. Zhang. Structured databases on the web: Observations and implications. *SIGMOD Record*, 33(3):61–70, 2004.

[7] V. Crescenzi and G. Mecca. Grammars have exceptions, 1998.

[8] V. Crescenzi, G. Mecca, P. Merialdo, et al. Roadrunner: Towards automatic data extraction from large web sites. In *Proceedings of the international conference on very large data bases*, pages 109–118, 2001.

[9] D. Embley, D. Campbell, Y. Jiang, S. Liddle, D. Lonsdale, Y.-K. Ng, and R. Smith. Conceptual-model-based data extraction from multiple-record web pages. 1999.

[10] F. A. For, M. E. Califf, and R. J. Mooney. Relational learning of pattern-match rules for information extraction. pages 328–334, 1999.

[11] D. Freitag. Information extraction from html: Application of a general machine learning approach. In *In Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 517–523, 1998.

[12] J. Hammer, J. Mchugh, and H. Garcia-molina. Semistructured data: The tsimmis experience. In *In Proceedings of the First East-European Workshop on Advances in Databases and Information Systems-ADBIS '97*, pages 1–8, 1997.

[13] A. Hogue and D. Karger. Thresher: Automating the unwrapping of semantic content from the world wide web. In *in Proceedings of the Fourteenth International World Wide Web Conference*, pages 86–95. ACM Press, 2005.

[14] C.-n. Hsu and M.-T. Dung. Generating finite-state transducers for semi-structured data extraction from the web, 1998.

[15] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[16] N. Kushmerick. Wrapper induction for information extraction, 1997.

[17] A. H. Laender, B. Ribeiro-Neto, and A. S. da Silva. {DEByE} data extraction by example. *Data & Knowledge Engineering*, 40(2):121 – 154, 2002.

[18] B. Liu, R. Grossman, and Y. Zhai. Mining data records in web pages. pages 601–606, 2003.

[19] B. Liu and Y. Zhai. Net a system for extracting web data from flat and nested data records. 2005.

[20] W. Liu, X. Meng, and W. Meng. Vide: A vision-based approach for deep web data extraction. *IEEE Transactions on Knowledge and Data Engineering*, 22(3), 2010.

[21] I. Muslea, S. Minton, and C. Knoblock. A hierarchical approach to wrapper induction. pages 190–197. ACM Press, 1999.

[22] M. ONeill, E. Hemberg, C. Gilligan, E. Bartley, J. McDermott, and A. Brabazon. Geva - grammatical evolution in java (v 1.0). Technical report, 2008.

[23] M. O'Neill, E. Hemberg, C. Gilligan, E. Bartley, J. McDermott, and A. Brabazon. Geva: grammatical evolution in java. *SIGEVOlution*, 3(2):17–22, July 2008.

[24] Pku-Hkust, J. Wang, and F. H. Lochvsky. Ode: Ontology-assisted data extraction. *IEEE Transactions on Database Systems*, 34(2), 2009.

[25] R. Poli, W. B. Langdon, and N. F. McPhee. *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd, 2008.

[26] B. Ribeiro-Neto, A. H. F. Laender, and A. S. D. Silva. Extracting semi-structured data through examples. In *In Proceedings of the International Conference on Knowledge Management*, pages 94–101, 1999.

[27] K. Simon and G. Lausen. Viper: augmenting automatic information extraction with visual perceptions. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, CIKM '05, pages 381–388, New York, NY, USA, 2005. ACM.

[28] S. Soderland, C. Cardie, and R. Mooney. Learning information extraction rules for semi-structured and free text. In *Machine Learning*, pages 233–272, 1999.

[29] J. Wang and F. H. Lochovsky. Data extraction and label assignment for web databases. In *Proceedings of the 12th international conference on World Wide Web*, pages 187–196. ACM, 2003.

[30] Y. Zhai and B. Liu. Web data extraction based on partial tree alignment. *ACM*, 2005.

[31] H. Zhao and W. Meng. Fully automatic wrapper generation for search engines. In *WWW Conference*, pages 66–75, 2005.