An Unsupervised Technique to Extract Information from Semi-structured Web Pages*

Hassan A. Sleiman and Rafael Corchuelo

University of Sevilla, Spain
{hassansleiman,corchu}@us.es

Abstract. We propose a technique that takes two or more web pages generated by the same server-side template and tries to learn a regular expression that represents it and helps extract relevant information from similar pages. Our experimental results on real-world web sites demonstrate that our technique outperforms others in terms of both effectiveness and efficiency and is not affected by HTML errors.

Keywords: Web information extraction, unsupervised learning.

1 Introduction

The Web is a huge information repository. Semi-structured web pages are generated by server-side scripts that retrieve information from databases and present it in HTML templates that introduce irrelevant information and attractive styles and layouts. Information extractors help extract the relevant information in a web page by using machine learning techniques.

Many information extractors rely on extraction rules. Although they can be handcrafted, the costs involved motivated many researchers to work on proposals to learn them automatically [4]. These proposals are either supervised, i.e., they require the user to provide a number of samples to be extracted, or unsupervised, i.e., they extract as much prospective information as they can and the user then gathers the relevant information from the results. Some authors have worked on unsupervised proposals that do not rely on extraction rules, but are based on a number of hypothesis and heuristics that have proven effective [1, 7]. Since typical web pages are growing in complexity, some authors are also paying attention to the problem of identifying information regions [11].

In this paper, we introduce a technique that allows to learn a regular expression that describes the structure of the template used to generate some input web pages; this expression can later be used to extract information from similar pages. The idea is to compare the input web pages in order to discover shared patterns that are common to all of them and, thus, are not likely to contain any

* Supported by the European Commission (FEDER), the Spanish and the Andalusian R&D&I programmes (TIN2007-64119, P07-TIC-2602, P08-TIC-4100, TIN2008-04718-E, TIN2010-21744, TIN2010-09809-E, TIN2010-10811-E, TIN2010-09988-E, and TIN2011-15497-E).

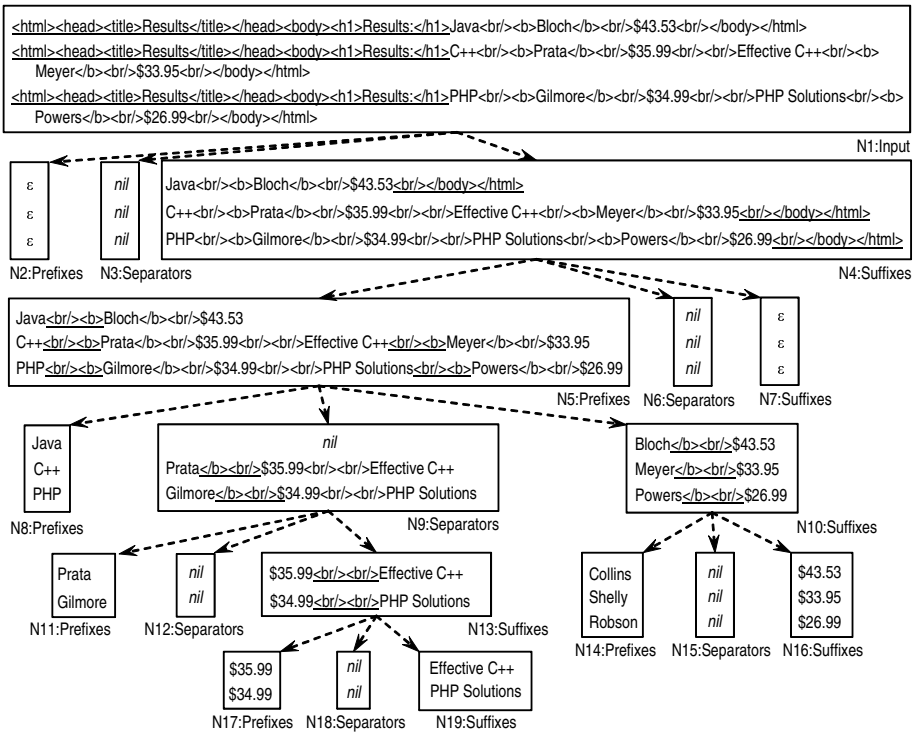

Fig. 1. A sample trinary tree. (Underlined strings represent shared patterns.)

relevant information. The idea of identifying shared patterns lies at the heart of the other related proposals, but the techniques they use differ significantly. EXALG [2] uses two statistical techniques to find the tokens that belong to the template; RoadRunner [5] uses a multi-string alignment algorithm that is exponential in the size of the input documents [6]; FiVaTech [8] relies on tree clustering and an ad-hoc matrix alignment technique. Our proposal relies on a multi-string alignment algorithm that has proven to be very effective and efficient in practice. Contrarily to the previous proposals, ours does not require the input web documents to be translated into DOM trees and thus does not require the input documents to be corrected so that they are well-formed HTML.

The rest of the paper is organised as follows: Section 2 describes the main algorithms of our proposal; Section 3 presents the results of our experimental evaluation; Section 4 draws a few conclusions; the paper finishes with some selected references to the literature.

2 Algorithm

Our algorithm works in two steps: first, it creates a trinary tree that represents the input pages and makes it explicit what fragments are shared patterns and the

```
createTrinaryTree(n: Node; min, max: nat)
    expanded = false
    size = max
    while size ≥ min and not expanded do
        expanded = expand(n, size)
        size = size - 1
    if expanded then
        leaves = getLeaves(n)
        foreach leaf in leaves do
            createTrinaryTree(leaf, min, size)
```

```
expand(n: Node; s: int): boolean
    result = false
    if size of n ≥ 2 then
        map, pattern = findPattern(n, s)
        if map ≠ {} then
            result = true
            set pattern of node to pattern
            foreach text in node do
                using map and text
                    create the prefixes of n
                    create the separators of n
                    create the suffixes of n
    return result
```

Fig. 2. Algorithms to create a trinary tree and to expand a node

prefixes, separators, and suffixes that they induce; later, this tree is transformed into a regular expression with capturing groups that represents the template used to generate the input pages. This expression allows to extract information from other pages that were generated by the same template thanks to the capturing groups.

Creating a trinary tree: Figure 1 presents a sample trinary tree that we use throughout the paper to illustrate our proposal. The tree is composed of Nodes of the form (T, a, p, e, s), where T is a collection of Texts, a is a Text that stores a shared pattern, and p, e, and s are three Nodes called prefixes, separators, and suffixes, respectively. A Text is a sequence of tokens, which represent HTML tags, script blocks, style blocks, and #PCDATA. We use ϵ to denote an empty Text and *nil* to refer to the inexistence of a Text or a Node.

Figure 2 presents our algorithm to create a trinary tree. It works on a Node with Texts that represent the input web pages and two naturals called *min* and *max* that limit the search for shared patterns to those of size *max* down to *min*. The algorithm first attempts to expand the input node, which is a process whose goal is to find a shared pattern in the collections of Texts in this node and use it to split them into new collections of prefixes, separators, and suffixes; if the input node is expanded, then the algorithm is applied recursively to the newly created leaves. To find shared patterns, we rely on Algorithm findPattern, which returns a map from Texts onto lists of naturals and a pattern; the former maps each Text in the input node onto the list of positions at which the latter was found. If the map is not empty, that implies that a shared pattern has been found, in which case, the algorithm sets the shared pattern of the input node to that pattern and then iterates over the Texts it contains and creates the corresponding prefix, separator, and suffix nodes.

To illustrate this algorithm, assume that it is invoked on Node $N1$ in Figure 1. It first searches for shared patterns in the Texts of this node and finds the following: <html><head><title>Results</title></head><body><h1>Results:

```
learnTemplate(n: Node; result: Regex): Regex
    if isOptional(n) then result += "("
    if isLeaf(n) then
        if not every texts in n is empty then
            result += "{" + freshLabel() + "}"
    else
        result += learnTemplate(prefix of n, result)
        result += pattern of n
        if isRepeatable(n, separators of n) then
            result += "(" + learnTemplate(separators of n, result) + getPattern(n)
            if nil is in separators of n then
                result += ")*"
            else
                result += ")+"
        result += learnTemplate(suffix of n, result)
    if isOptional(n) then result += ")?"
    return result
```

Fig. 3. Algorithm to learn a template from a trinary tree

</h1>. It then expands Node $N1$ into three additional nodes, namely: $N2$, $N3$, and $N4$. Since the shared pattern is found at the beginning of the Texts in $N1$ and it is not repeated in any of them, then Node $N2$, which contains the prefixes of the shared pattern, only contains three empty Texts; node $N3$, which contains the separators between the occurrences of the shared pattern in each Text, only contains three *nil* values since there are not any separators; contrarily, there are three suffixes that are stored in Node $N4$. Then, the algorithm is applied recursively to $N2$, $N3$, and $N4$ to search for new shared patterns. $N2$ and $N3$ are not processed again since they only contain empty or *nil* Texts; only Node $N4$, whose Texts share the 3-token pattern
</body></html>, is expanded again to create Nodes $N5$, $N6$, and $N7$. (Note that there are other shared patterns, but the algorithm searches for the longest one and breaks ties arbitrarily.) The same procedure is applied as many times as necessary until no more shared patterns are discovered.

Learning a regular expression: Figure 3 presents our algorithm to learn a regular expression from a trinary tree. It relies on two ancillary concepts: optionality and repeatability. A Node is optional if one or more of its Texts, but not all, are empty; a Node is repeatable if one or more of its non-empty Texts have more than one occurrence of the shared pattern, which implies that the separators Node contains one or more Texts that are not equal to *nil*.

The algorithm proceeds as follows: it works on a node and a regex that is expected to be an empty string initially; the algorithm constructs its result by adding text to this parameter on each recursive invocation. If the node being processed is a leaf and not all of its Texts are empty, the algorithm then adds a capturing group that represents a piece of text that has to be extracted; if it

is not a leaf, then the algorithm builds recursively the regular expressions that correspond to the prefixes, the separators, and the suffixes. If the separators node is repeatable, then the decision on whether to use a star or a plus closure is made as follows: if *nil* is included in the Texts in the separators node, then it means that there is at least an input web page in which the separator does not appear, in which case, a star closure must be used; otherwise, a plus closure must be used. The first and the last lines of the algorithm take into account the case in which the node being processes is optional; in such cases, a parenthesis and an optional operator are added to the resulting regular expression.

The template learnt for our running example is the following:

```
(<html><head><title>Results</title></head><body><h1>Results:</h1>) {_A_} (<br/><b>)
( ({_B_} (</b><br/>) {_C_} (<br/><br/>) {_D_} )? (<br/><b>) )*
{_E_} (</b><br/>) {_F_} (<br/></body></html>)
```

As is the case in other unsupervised techniques, it is the user who must assign a meaning to the capturing groups. In our running example, _A_ and _D_ stand for titles, _B_ and _E_ stand for authors, and _C_ and _F_ stand for prices. The problem of mapping the information extracted by the capturing groups onto structured records was dealt with elsewhere [3].

3 Experimentation

We have developed a prototype of our proposal using the CEDAR framework [10]. We performed a series of experiments on a cloud computer that was equipped with a four-threaded Intel Core i7 processor that ran at 2.93 GHz, had 4 GB of RAM, Windows 7 Pro 64-bit, and Oracle's Java Development Kit 1.7.0_02. The default heap size of the Java Virtual Machine was not modified. The experiments were carried out on a collection of 29 datasets that provide 688 web pages. The datasets were gathered from real-world web sites on books, doctors, movies, and from the EXALG repository [2]. We experimentally found that setting $min = 1$ and $max = \lfloor 0.05m \rfloor$ was the maximum allowable bias to our search procedure, where m denotes the size of the longest input document; this resulted in a significant increase in efficiency without an impact on effectiveness.

We ran our proposal, RoadRunner [5], FiVaTech [8], and WIEN [9] on the datasets in order to learn extraction rules. (Unfortunately, we could not find a public implementation of EXALG, so we added WIEN to our comparison since it ranks amongst the most cited proposals in information extraction.) We then computed the usual effectiveness and efficiency measures. In the case of WIEN, it was easy to compute the effectiveness measures since the technique is supervised, i.e., it requires the user to provide annotations with the information to be extracted so that an extraction rule can be learnt and evaluated. Contrarily, our proposal, RoadRunner, and FiVaTech are unsupervised, i.e., they learn an extraction rule that extracts as much information as possible, give computer-generated labels to the capturing groups, and it is the responsibility of the user to assign a meaning to these labels. We then used the following approach: we compared the information extracted by each capturing group to every annotation

Table 1. Results of our experimentation

				Trinity				RoadRunner				FivaTech				WIEN			
	Summary	N	E	P	R	LT	ET	P	R	LT	ET	P	R	LT	ET	P	R	LT	ET
	Mean	28.67	37.63	0.98	0.95	0.13	0.01	0.49	0.49	44.40	0.01	0.81	0.92	116.18	0.24	0.68	0.55	7.70	9.83
	Standard deviation	8.60	37.98	0.03	0.11	0.18	0.02	0.47	0.46	193.77	0.01	0.19	0.11	192.06	0.44	0.24	0.30	4.93	6.79
	Site	N	E	P	R	LT	ET	P	R	LT	ET	P	R	LT	ET	P	R	LT	ET
Books	www.abebooks.com	30	2.94	1.00	1.00	0.03	0.00	-	-	-	-	0.92	0.99	15.46	0.12	0.52	0.16	9.66	10.26
Books	www.awesomebooks.com	30	2.16	1.00	0.87	0.03	0.00	1.00	1.00	0.92	0.00	0.85	1.00	8.14	0.14	0.77	0.26	5.01	6.27
Books	www.betterworldbooks.com	30	2.30	0.99	1.00	0.17	0.00	0.00	0.00	0.98	0.00	0.99	0.96	85.32	0.39	0.43	0.35	15.57	17.04
Books	www.manybooks.net	30	6.50	0.99	0.99	0.11	0.02	-	-	-	-	0.77	0.97	65.49	0.12	0.25	0.23	6.74	8.14
Books	www.waterstones.com	30	6.46	0.96	1.00	0.07	0.00	1.00	0.89	1.14	0.02	1.00	0.94	51.53	1.98	0.71	0.67	8.02	8.72
Doctors	doctor.webmd.com	30	24.10	1.00	1.00	0.02	0.00	0.00	0.00	0.73	0.00	0.77	1.00	11.81	0.03	0.60	0.60	9.45	14.54
Doctors	extapps.ama-assn.org	30	36.00	0.98	1.00	0.02	0.02	-	-	-	-	-	-	-	-	0.60	0.60	6.99	7.38
Doctors	www.dentists.com	30	103.27	0.92	1.00	0.01	0.00	1.00	1.00	867.63	0.00	0.56	0.99	9.94	0.08	1.00	1.00	1.97	1.84
Doctors	www.drscore.com	30	33.07	1.00	1.00	0.03	0.00	1.00	1.00	3.28	0.02	0.78	1.00	25.35	0.06	0.78	0.80	3.88	3.62
Doctors	www.steadyhealth.com	30	24.00	1.00	1.00	0.67	0.02	0.00	0.00	0.67	0.02	0.83	0.83	9.59	0.11	0.75	0.75	9.56	9.77
Movies	www.allmovie.com	30	59.40	0.97	0.96	0.21	0.00	0.27	0.30	1.64	0.03	0.79	0.74	14.84	0.11	0.13	0.07	7.78	5.91
Movies	www.citwf.com	30	20.90	1.00	1.00	0.02	0.00	1.00	1.00	0.69	0.02	1.00	1.00	29.70	0.05	0.39	0.30	2.79	3.79
Movies	www.disneymovieslist.com	30	32.33	1.00	1.00	0.07	0.00	0.00	0.00	0.59	0.00	0.71	0.67	259.23	0.08	0.72	0.72	3.95	3.82
Movies	www.imdb.com	30	138.80	0.93	0.86	0.45	0.03	0.00	0.00	0.97	0.02	-	-	-	-	0.38	0.38	15.87	16.46
Movies	www.soulfilms.com	30	66.13	0.99	0.92	0.03	0.00	0.00	0.00	0.47	0.03	0.59	1.00	17.24	0.05	0.91	0.81	9.73	9.36
EXALG	cars.amazon.com	21	20.00	0.93	0.73	0.01	0.00	0.27	0.33	0.55	0.02	0.60	0.67	2.29	0.05	0.97	1.00	19.47	8.10
EXALG	players.uefa.com	20	10.40	1.00	0.90	0.02	0.02	0.92	0.92	0.51	0.02	0.91	0.94	10.81	0.05	0.92	0.51	3.53	3.70
EXALG	popartist.amazon.com	19	35.00	1.00	0.98	0.02	0.00	0.99	0.99	0.98	0.03	1.00	1.00	246.97	0.11	0.92	0.58	15.66	10.22
EXALG	teams.uefa.com	20	32.80	0.99	0.99	0.03	0.00	0.90	0.92	0.28	0.02	0.97	0.99	0.89	0.02	0.64	0.75	1.79	2.28
EXALG	www.ausopen.com	29	66.73	1.00	1.00	0.26	0.02	0.37	0.39	1.15	0.02	0.24	0.82	132.24	0.27	0.67	0.32	9.67	16.96
EXALG	www.ebay.com	50	18.12	0.97	1.00	0.51	0.08	0.00	0.00	2.51	0.02	0.83	1.00	577.97	0.48	0.70	0.12	5.44	19.55
EXALG	www.majorleaguebaseball.com	9	26.00	0.98	0.55	0.04	0.00	0.00	0.00	0.95	0.02	0.99	1.00	158.33	0.06	0.65	0.33	3.60	6.13
EXALG	www.netflix.com	50	125.86	0.99	0.99	0.18	0.02	-	-	-	-	0.82	0.80	706.64	0.76	0.99	0.99	7.47	31.54
EXALG	www.rpmfind.net	20	9.90	0.95	0.97	0.03	0.05	0.98	0.99	1.31	0.03	-	-	-	-	0.99	0.99	1.29	10.42

Table 2. Correlation from number of errors to effectiveness

	Trinity		RoadRunner		FiVaTech		WIEN	
	P	R	P	R	P	R	P	R
Coefficient	-0.14	-0.08	-0.11	-0.06	-0.33	-0.08	0.08	0.20
P-value	0.38	0.61	0.04	0.74	0.04	0.64	0.57	0.16

and computed the precision and recall. Then, we considered that the precision and recall of the extracted information corresponds to the extracted piece of text with the highest harmonic mean of precision and recall, i.e., the F_1 measure.

Table 1 shows our results. The first few rows provide a summary in terms of mean and standard deviations of the number of web pages in each dataset (N), the average number of errors in each dataset (E), precision (P), recall (R), rule learning time in CPU seconds (LR), and extraction time in CPU seconds (ET). The remaining rows provide the results we computed for each web site. (A dash, which indicates that the corresponding technique was not able to learn an extraction rule in 15 CPU minutes.) In average, our proposal seems to outperform the other techniques in both precision and recall, with a mean learning and extraction times that are clearly smaller than the other techniques'; the only exception is RoadRunner, whose extraction time seems similar to ours.

We were also interested in determining if there was a correlation from the number of errors in a collection of input pages to the effectiveness of our proposal. Unfortunately, it is not easy to draw an intuitive conclusion from the

data in the table. We then conducted a statistical analysis of correlation using the well-known non-parametric Kendall's τ procedure. Table 2 shows the results of the study. Note that there are only two p-values that are smaller than the standard significance level $\alpha = 0.05$: the one that corresponds to the precision of RoadRunner and the one that corresponds to the precision of FiVaTech; this implies that the number of errors in the input web pages seems to have an impact on the precision of these techniques, whereas the impact on our proposal is not significant from a statistical point of view.

4 Conclusions

We have proposed a new and effective unsupervised information extractor that is based on the hypothesis that web pages generated by the same server-side template share patterns that provide irrelevant information. The rule learning algorithm searches for these patterns, builds a trinary tree, which is then used to learn a regular expression that represents the template that was used to generate the input web pages. Our experiments on real-world web pages proved that our technique is highly effective and efficient and that it is not significantly influenced by the presence of errors in the input HTML pages.

References

[1] Álvarez, M., Pan, A., Raposo, J., Bellas, F., Cacheda, F.: Extracting lists of data records from semi-structured web pages. Data Knowl. Eng. 64(2), 491–509 (2008)

[2] Arasu, A., Garcia-Molina, H.: Extracting structured data from web pages. In: SIGMOD Conference, pp. 337–348 (2003)

[3] Arjona, J.L., Corchuelo, R., Ruiz, D., Toro, M.: From wrapping to knowledge. IEEE Trans. Knowl. Data Eng. 19(2), 310–323 (2007)

[4] Chang, C.H., Kayed, M., Girgis, M.R., Shaalan, K.F.: A survey of web information extraction systems. IEEE Trans. Knowl. Data Eng. 18(10), 1411–1428 (2006)

[5] Crescenzi, V., Mecca, G.: Automatic information extraction from large websites. J. ACM 51(5), 731–779 (2004)

[6] Crescenzi, V., Mecca, G., Merialdo, P.: RoadRunner: Towards automatic data extraction from large web sites. In: VLDB, pp. 109–118 (2001)

[7] Elmeleegy, H., Madhavan, J., Halevy, A.Y.: Harvesting relational tables from lists on the Web. PVLDB 2(1), 1078–1089 (2009)

[8] Kayed, M., Chang, C.H.: FiVaTech: Page-level web data extraction from template pages. IEEE Trans. Knowl. Data Eng. 22(2), 249–263 (2010)

[9] Kushmerick, N., Weld, D.S., Doorenbos, R.B.: Wrapper induction for information extraction. In: IJCAI (1). pp. 729–737 (1997)

[10] Sleiman, H.A., Corchuelo, R.: A Reference Architecture to Devise Web Information Extractors. In: Bajec, M., Eder, J. (eds.) CAiSE Workshops 2012. Lecture Notes in Business Information Processing, vol. 112, pp. 235–248. Springer, Heidelberg (2012)

[11] Sleiman, H.A., Corchuelo, R.: A survey on region extractors from web documents. IEEE Trans. Knowl. Data Eng. 99(pre-prints) (2012)