Trinity: On Using Trinary Trees for Unsupervised Web Data Extraction

Hassan A. Sleiman and Rafael Corchuelo

Abstract—Web data extractors are used to extract data from web documents in order to feed automated processes. In this article, we propose a technique that works on two or more web documents generated by the same server-side template and learns a regular expression that models it and can later be used to extract data from similar documents. The technique builds on the hypothesis that the template introduces some shared patterns that do not provide any relevant data and can thus be ignored. We have evaluated and compared our technique to others in the literature on a large collection of web documents; our results demonstrate that our proposal performs better than the others and that input errors do not have a negative impact on its effectiveness; furthermore, its efficiency can be easily boosted by means of a couple of parameters, without sacrificing its effectiveness.

Index Terms—Web data extraction, automatic wrapper generation, wrappers, unsupervised learning

1 Introduction

The Web is a huge repository in which data are usually presented using friendly formats, which makes it difficult for automated processes to use them.

The literature provides many proposals to create so-called web data extractors, which are tools that facilitate extracting relevant data from typical web documents [9], [35]. Many web data extractors rely on extraction rules, which can be classified into ad-hoc or built-in rules. The costs involved in handcrafting ad-hoc rules motivated many researchers to work on proposals to learn them automatically using supervised techniques, i.e., techniques that require the user to provide samples of the data to be extracted, aka annotations [4]–[7], [10], [15]–[17], [20], [23], [24], [27], [29], [37], [38], or using unsupervised techniques, i.e., techniques that learn rules that extract as much prospective data as they can, and the user then gathers the relevant data from the results [2], [8], [11], [19], [21], [25], [26], [38]–[40]. Web data extractors that rely on built-in rules are based on a collection of heuristic rules that have proven to work well on many typical web documents [1], [14], [18], [32], [36]. Since such documents are growing in complexity, some authors are also working on techniques whose goal is to identify the region within a web document where the relevant data is most likely to reside [35]. Some authors have also paid attention to the problem of structuring the data extracted [3], [30].

In this article, we introduce a technique called Trinity, which is an unsupervised proposal that learns extraction rules from a set of web documents that were generated by the same server-side template. It builds on the hypothesis that shared patterns are not likely to provide any relevant data and are, thus, part of the template. Whenever it finds a shared pattern, it partitions the input documents into the prefixes, separators and suffixes that they induce and analyses the results recursively, until no more shared patterns are found. Prefixes, separators, and suffixes are organised into a trinary tree that is later traversed to build a regular expression with capturing groups that represents the template that was used to generate the input documents. Thanks to the capturing groups, the expression can be used to extract data from similar documents. Note that our technique does not require the user to provide any annotations; instead, he or she must interpret the resulting regular expression and map the capturing groups that represent the information of interest onto the appropriate structures.

The three most closely-related proposals are RoadRunner [11]–[13], ExAlg [2], and FiVaTech [21], which differ significantly from Trinity: RoadRunner is a parsing-based approach that uses a partial rule (which is initialised to any of the input documents) to parse another document and applies a number of generalisation strategies to correct the partial rule when mismatches are found; ExAlg finds maximal classes of tokens that occur in every input document, which are thus very likely to belong to the template, and then refines them using a token differentiation and a nesting criteria in order to construct the extraction rule; FiVaTech first identifies nodes in the input DOM trees that have a similar structure and then aligns their children and mines repetitive and optional patterns to create the extraction rule.

We analysed the complexity of our proposal and proved that it is polynomial in both space and time. We also conducted a series of experiments with 55 real-world web sites and our results confirm that our proposal can achieve a mean precision as high as 0.96 ± 0.07, a mean recall as high as 0.95 ± 0.11, with a mean learning time of 0.13 ± 0.16

• *The authors are with the University of Sevilla, ETSI Informática, Sevilla E-41012, Spain. E-mail: {hassansleiman, corchu}@us.es.*

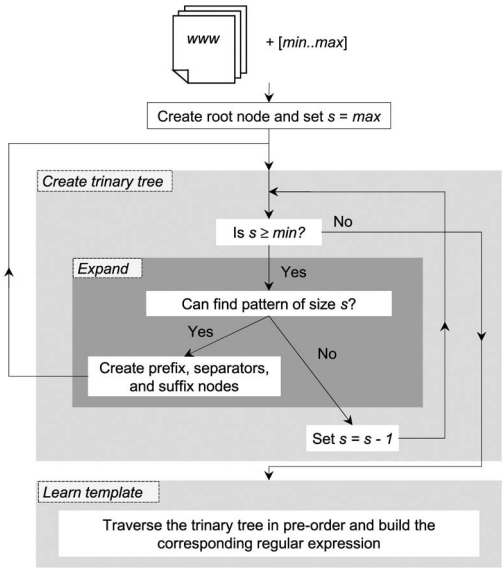


Fig. 1. General view of our proposal.

CPU seconds and a mean extraction time of 0.02 ± 0.03 CPU seconds. We conducted the same experiments using two of the most closely-related techniques available and two well-known supervised proposals in the literature, and we then analysed the results using sound statistical techniques. Our conclusion was that our proposal performs better than the others and that its effectiveness does not depend at all on whether the input web documents are well- or malformed. Other techniques require the input to be correct XHTML, which requires to repair the source documents beforehand; experimentally, we found out that this has a negative impact on their effectiveness. Our proposal relies on two simple parameters that may introduce a bias in order to boost its performance, without sacrificing its effectiveness. Other techniques have parameters that need to be fined-tuned so that they work well.

The rest of the article is organised as follows: Section 2 presents the algorithms that lie at the heart of Trinity; Section 3 analyses both their space and time complexity; Section 4 reports on our experimental results; Section 5 compares it to the most closely-related proposals from a conceptual point of view; Section 6 concludes our work. A short paper on Trinity was presented elsewhere [34].

2 Description of our Algorithms

In this section, we first provide an intuitive introduction to our proposal, then report on its limitations, and finally provide an algorithmic description.

2.1 Overview

Fig. 1 sketches our proposal, which takes a collection of web documents and a natural range $[min \,..\, max]$ as input. The web documents need to be tokenised, but they do not need to be correct XHTML documents; the range indicates the minimum and maximum size of the shared patterns for which the algorithm searches.

Our proposal relies on the following data types: a sequence of tokens is called Text and represents either a whole input document or a fragment; a trinary tree is a collection of Nodes, each of which is a tuple of the form (T, a, p, e, s), where T is a collection of Text, a is of type Text and contains a shared pattern in T, p is a Node called prefixes, e is a Node called separators, and s is a Node called suffixes.

The algorithm first creates a root node with the input web documents and sets a variable called s to max. Starting with this node, the algorithm loops and searches for a shared pattern of size s. If such a pattern is found in the current node, then it is used to create three new child nodes with the prefixes, the separators, and the suffixes that this pattern induces; the prefixes are the fragments from the beginning of a Text up to the first occurrence of a shared pattern, the separators are the fragments in between successive occurrences, and the suffixes are the fragments from the last occurrence until the end of a Text. These nodes are analysed recursively in order to find new shared patterns that induce new nodes. If no shared pattern is found, that is, the tree is not expanded, but variable s is greater or equal to the minimum pattern size, then s is decreased and the procedure is repeated again until a node in which no shared pattern of size greater or equal to min is found.

Fig. 2 shows a sample trinary tree. Node $N1$ represents three sample input documents; the algorithm searches for the longest shared pattern, which is underlined, and then creates three new nodes with the prefixes, the separators, and the suffixes that it induces. Note that the shared pattern is found at the beginning of the input documents, so the prefixes are empty strings that we represent using symbol ε; note that the shared pattern occurs only once, which implies that there are not actually any separators, which we represent with symbol nil. The processing continues recursively on the new nodes. Note that leaf nodes that have variability, that is, contain different fragments, have data that is not likely at all to belong to the template.

Once the trinary tree is built, we need to use an additional algorithm to learn the regular expression that represents the template used to generate the input web documents. This algorithm traverses the trinary tree in pre-order; every time it reaches a leaf node that has variability, it outputs a fresh capturing group to extract the data that corresponds to that node; otherwise, it outputs the shared pattern that corresponds to the node being analysed and a closure or an optional operator depending on whether that node is repeatable (there can be multiple occurrences of the shared pattern) or optional (if some of the Texts it has are empty, but not all of them). Fig. 3(a) shows the regular expression that Trinity learns for the sample trinary tree in Fig. 2; capturing groups are represented as {A}, {B}, ..., {F}.

2.2 Limitations

Note that when the input documents have listings of records of different lengths, Trinity tends to deal with the first few attributes of the first record and the last few attributes of the last record differently from the attributes of the remaining records. For instance, in our illustrating example, the algorithm works on a collection of three web documents that have lists of book records, each of which has three attributes: title, authors, and price. Note, however,

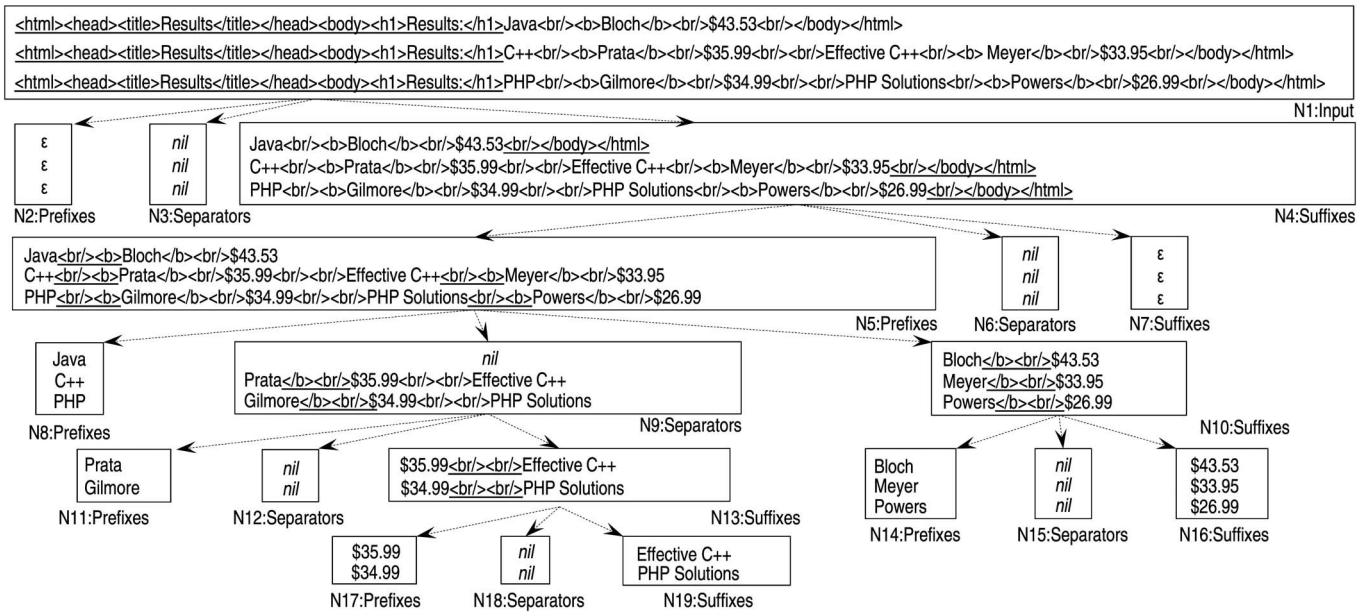


Fig. 2. Sample trinary tree. (Shared patterns are underlined in each node.)

that the regular expression that Trinity learns has six capturing groups: A corresponds to the title of the first record, E and F correspond to the authors and price of the last record; B, C, and D correspond to the remaining authors, prices, and titles, respectively. This limitation is not so uncommon in other proposals; fortunately, some authors have worked on techniques that allow to map the data returned by extraction rules onto more appropriate data structures [3], [30]. We have, however, found quite a simpler approach to overcome this limitation: Trinity is an unsupervised technique, which means that it is the user who has to assign a semantic label to each capturing group; after that, several computer-generated labels may be assigned to the same semantic label, which results in a regular expression that can be very easily simplified as follows: transform it into a deterministic finite automata, then minimise the automata, and transform the results back into a regular expression. This results in the regular expression in Fig. 3(b), which represents faithfully the structure of the records in the input documents.

Trinity learns union-free regular expressions. On a first reading, this might seem to prevent our proposal from working well with templates that have alternating formattings for the same data. Although this is true, we did not find this a serious limitation in practice because typical web documents that use alternating formattings rely on CSS classes. For instance, a special price is typically output as follows: $99.95; and a normal price is output as follows: $125.00. That is, the internal structure is almost identical although the rendering seems to suggest that it is not. There are cases in which the alternating formatting relies on tags like strong, emph, or strike, which are output or not depending on whether a piece of data is going to be highlighted or not. Since we do not rely on a fixed tokenisation scheme, we may easily deal with such tags as if they were regular text. This is the same approach that was used in the implementation of RoadRunner and it has proved to work quite well in practice.

Fig. 3. Regular expression learnt from our sample tree: (a) Before simplification. (b) After simplification.

An additional limitation occurs when the same sequence of tokens is used to separate different attributes in a data record. For instance, assume that data attributes are separated only by tag
; in such cases, Trinity is likely to find that the data record includes multiple instances of the same attribute. This limitation was not usual at all in the datasets we used to evaluate our proposal; we only had trouble with the Major League and the IAF datasets, in which data were rendered in a simple table and we achieved an $F1$ measure of 0.70 and 0.52, respectively. In the rest of datasets, different formattings are used inside the data records to render different attributes, and this helps our technique extract them independently. This limitation was also detected in ExAlg [2]; the authors suggested that human effort and more HTML knowledge are necessary in this case.

2.3 Creating a Trinary Tree

We present Algorithm createTrinaryTree in Fig. 4. It takes a node and a range of naturals as input and expands the node to create a trinary tree. The loop at lines 4-7 iterates over every possible size from *max* down to *min* as long as Algorithm expand cannot expand the current node,

```
 1: createTrinaryTree(node: Node; min, max: nat)
 2:    expanded = false
 3:    size = max
 4:    while size ≥ min and not expanded do
 5:       expanded = expand(node, size)
 6:       size = size − 1
 7:    end
 8:    if expanded then
 9:       foreach leaf in the leaves of node do
10:          createTrinaryTree(leaf, min, size + 1)
11:       end
12:    end
13: end
```

Fig. 4. Algorithm createTrinaryTree.

which happens if it can find a shared pattern of the indicated size. If the loop finishes and the current node was expanded, then the algorithm is executed recursively on the new leaves that have been added to the current node.

We present Algorithm expand in Fig. 5. Line 3 checks if *node* contains more than one Text; if so, it searches for a shared pattern by invoking Algorithm findPattern at line 4, which is described below; the algorithm checks if findPattern has found a shared pattern at line 5 and invokes Algorithm createChildren at line 7. Algorithm createChildren, which is described below, partitions the Text collection inside the input Node, and creates its child nodes; if no shared pattern is found, then the input Node remains unchanged.

The previous algorithms were relatively intuitive. The key to their performance is the algorithm to find shared patterns, which we present in Fig. 6. First, it searches for the shortest non-empty Text inside *node* at line 3 and stores it in *base*. The main loop at lines 5-16 allows to implement a sliding window over *base*: index i iterates from 0 until $size(base) - s$ as long as no shared pattern is found. The actual search is performed in the inner loop at lines 8-12: in this loop, the algorithm iterates over every Text in the input Node, and finds all of the matches of the subsequence of *base* that starts at position i and has size s. Algorithm findMatches is implemented using the well-known Knuth-Pratt-Morris pattern search algorithm [22]. This algorithm returns a list of naturals that indicate the non-overlapping positions at which the previous subsequence of *base* matches *text*; if there is at least one match, we record it in variable *result* and go ahead to examine the next Text in *node*; otherwise, the inner loop finishes and the outer loop slides the window on *base* and resets *map*, if possible. If the algorithm returns an empty map, this means that no shared pattern has been found; otherwise, the map indicates the positions at which the shared pattern is found in each Text in the node being analysed.

```
 1: expand(node: Node; s: nat): boolean
 2:    result = false
 3:    if size(node) > 1 then
 4:       map, pattern = findPattern(node, s)
 5:       if map ≠ {} then
 6:          result = true
 7:          createChildren(node, map, pattern)
 8:       end
 9:    end
10: return result
```

Fig. 5. Algorithm expand.

```
 1: findPattern(node: Node; s: nat): Map⟨Text, List⟨nat⟩⟩, Text
 2:    found = false
 3:    base = findShortestText(node)
 4:    pattern = ⟨⟩
 5:    for i = 0 until size(base) - s while not found do
 6:       map = {}
 7:       found = true
 8:       foreach non-empty text in node while found do
 9:          matches = findMatches(text, base, i, s)
10:          found = size(matches) > 0
11:          map = map ∪ {text ↦ matches}
12:       end
13:       if found then
14:          pattern = subsequence(base, i, s)
15:       end
16:    end
17: return map, pattern
```

Fig. 6. Algorithm findPattern.

When a shared pattern is found in a Node, Algorithm createChildren creates the new child nodes: prefixes, separators, and suffixes. We present Algorithm createChildren in Fig. 7. This algorithm works on a Node, a map, and a pattern. Lines 3-5 create three empty Nodes, namely *prefixes*, *separators*, and *suffixes*. Line 6 sets the node's shared pattern to *pattern*. Then, the loop at lines 7-12 iterates over the Texts in map: for each Text *text* in the map, lines 9-11 get the matches where the shared pattern occurs, computes the prefix, separators, and suffix of the pattern in *text*, and adds them to the *prefixes*, *separators*, and *suffixes* Nodes respectively. If *pattern* is found at the beginning of *text*, the prefix is then an empty Text; if *pattern* is found at the end of *text*, the suffix is then an empty Text; if two occurrences of *pattern* are consecutive in *text*, their separator is an empty Text, but if *text* contains only one occurrence of *pattern*, we then add the special value *nil* to *separators*. We do not provide a pseudocode to the algorithms to compute prefixes, separators, and suffixes since they are quite straightforward.

```
 1: createChildren(node: Node; map: Map⟨Text, List⟨nat⟩⟩;
 2:                pattern: Text)
 3:    prefixes = new Node()
 4:    separators = new Node()
 5:    suffixes = new Node()
 6:    set pattern of node to pattern
 7:    foreach text in node do
 8:       matches = map(text)
 9:       add computePrefix(matches, text) to prefixes
10:       add computeSeparators(matches, text) to separators
11:       add computeSuffix(matches, text) to suffixes
12:    end
13:    set prefix of node to prefixes
14:    set separators of node to separators
15:    set suffix of node to suffixes
16: end
```

Fig. 7. Algorithm createChildren.

```
 1: learnTemplate(node: Node; result: Regex): Regex
 2:     if isOptional(node) then result += "(" end
 3:     if node is a leaf then
 4:         if node has variability then
 5:             result += freshCapturingGroup()
 6:         end
 7:     else
 8:         result += learnTemplate(prefix of node, result)
 9:         result += pattern of node
10:         if isRepeatable(node, separators of node) then
11:             result += "(" +
12:                 learnTemplate(separators of node, result) +
13:                 pattern of node
14:             if contains(separators of node, nil) then
15:                 result += ")*"
16:             else
17:                 result += ")+"
18:             end
19:         end
20:         result += learnTemplate(suffix of node, result)
21:     end
22:     if isOptional(node) then result += ")?" end
23: return result
```

Fig. 8. Algorithm learnTemplate.

2.4 Algorithm learnTemplate

Algorithm learnTemplate works on a trinary tree whose root is *node*, constructs a regular expression that represents a template, and returns it, cf. Fig. 8. It relies on two ancillary algorithms, namely: isOptional and isRepeatable. 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.

The algorithm proceeds as follows: it works on a node and a regular expression that is expected to be empty initially; the algorithm constructs its result by adding text to this parameter on each recursive invocation. The core is the if-then-else sentence at lines 3-31, which distinguishes between leaf and non-leaf nodes. If *node* is a leaf and has variability, i.e., not all of its Texts are the same, this means that it contains variable data. In such cases, line 5 creates a new capturing group that represents a piece of text to be extracted. If *node* is not a leaf, then line 8 builds the regular expression that corresponds to the prefixes, which is performed recursively. The shared pattern is added to *result* at line 9 and the regular expression that corresponds to the separators is built at lines 10-18: if the node is repeatable, then the regular expression of the separators Node is built at line 12, the shared pattern is added to *result* at line 13, and the plus or star closures are added at lines 14-18. If the separators node contains the special value *nil*, this means that the shared pattern has appeared only once in at least one of the Texts in *node*, thus a star closure must be used; contrarily, if the shared pattern has two or more occurrences in each Text in *node*, then a plus closure must be added. The algorithm builds now the regular expression that corresponds to the suffixes at line 20, which is performed recursively.

Lines 2 and 22 check if the node being processed is optional, in which case parenthesis and an optional operator are added to the resulting regular expression.

3 Complexity Analysis

In this section, we first present the results of our complexity analysis and then prove the lemmata that support them regarding space and time requirements.

3.1 Complexity Results

Let n denote the number of documents that Trinity has to analyse and let m denote the size in tokens of the longest such document. From Lemmata 1 and 2, which we present below, we conclude that $n\lfloor\frac{m}{2}\rfloor$ is an upper bound to the maximum size of a node and that $3m$ is an upper bound to the maximum number of nodes the algorithm creates; as a conclusion, $O(n\lfloor\frac{m}{2}\rfloor 3m) \subseteq O(n\,m^2)$ is an upper bound to the space required to execute our proposal.

Furthermore, according to Lemmata 6 and 7, which we present below, Algorithms createTrinaryTree and learnTemplate require no more than $O(n\,m^5)$ and $O(n\,m^2)$ time to complete, respectively. This implies that $O(n\,m^5 + n\,m^2) \subseteq O(n\,m^5)$ is an upper bound to the worst-case time required to execute our proposal.

In our experiments, it was common that $n \ll m$ since the average value of n was 37.89 ± 41.67 and the average value of m was $1\,842.40 \pm 1\,445.31$; in such cases, we can conclude that $O(m^2)$ and $O(m^5)$ are upper bounds to the worst-case space and time complexity of Trinity. That is: our proposal is computationally tractable.

3.2 Space Requirements

Lemma 1 (Maximum size of a Node). *$n\lfloor\frac{m}{2}\rfloor$ is an upper bound to maximum size of a Node created by Algorithm createTrinaryTree.*

Proof. Algorithm expand is the only algorithm that creates new Nodes, which correspond to the prefixes, separators, and suffixes that a shared pattern p induces. Regarding the prefix and suffix Nodes, note that there cannot be more than n such prefixes or suffixes since a document may not have more than one prefix or one suffix; that is, n is an upper bound to the maximum size of a Node that contains prefixes or suffixes. Regarding separators Nodes the worst-case happens when p is a one-token pattern that occurs every two tokens; that is, $\lfloor\frac{m}{2}\rfloor$ is an upper bound to the number of separators in this case, that is, $n\lfloor\frac{m}{2}\rfloor$ is an upper bound to the maximum size of a Node that contains separators. As a conclusion, $n\lfloor\frac{m}{2}\rfloor$ is an upper bound to the maximum size of a Node created by createTrinaryTree. □

Lemma 2 (Maximum number of Nodes). *$3m$ is an upper bound to the number of Nodes created by Algorithm createTrinaryTree.*

Proof. Algorithm expand creates three new Nodes when a shared pattern p is found in a given Node. The new Nodes correspond to the prefixes, separators, and suffixes to which p leads. We know that m is an upper bound to the number of partitions of a Text of size m, which means that m is an upper bound to the number of levels of the trinary tree in the worst-case. Since each level has three nodes, then $3m$ is an upper bound to the number of Nodes created by createTrinaryTree. □

3.3 Time Requirements

Lemma 3 (Algorithm createChildren). *Let a be a Node, r a map from the Texts in a onto lists of indices that denote where a shared pattern occurs, and p the shared pattern. $O(n\,m^2)$ is an upper bound to the worst-case time required to execute createChildren(a, r, p).*

Proof. The algorithm iterates through every Text in a. According to Lemma 1, $n\lfloor\frac{m}{2}\rfloor$ is an upper bound to the maximum size of a Node, which means that the $n\,m$ is an upper bound to the number of iterations of this loop. Inside this loop, accessing the map and calculating the prefix and suffix of the shared pattern can be performed in $O(1)$ time, whereas computing the separators requires variable time. According to Lemma 1, the maximum number of separators in a given Text is $\lfloor\frac{m}{2}\rfloor$, which is less than m. Then, $O(m)$ is an upper bound to the time required to compute the separators. The instructions to create new nodes at lines 3-5 and the instructions to link them to the input Node at lines 13-15 require $O(1)$ time. As a conclusion, $O(n\,m\,m) = O(n\,m^2)$ is an upper bound to the worst-case time required to execute createChildren(a, r, p). □

Lemma 4 (Algorithm findPattern). *Let a be a Node and s the size of the pattern for which the algorithm searches. $O(n\,m^3)$ is an upper bound to the worst-case time required to execute findPattern(a, s).*

Proof. The algorithm first searches for the shortest Text in a. According to Lemma 1, $n\lfloor\frac{m}{2}\rfloor$ is an upper bound to the maximum size of a Node, which implies that $n\lfloor\frac{m}{2}\rfloor$ is an upper bound to the maximum time required to find the shortest Text in a Node. The main loop iterates through *base* until finding a pattern of s tokens that occurs in every other Text in a. In the worst-case, *base* has the maximum size m and the shared pattern is found at the end of *base*, which means that the main loop iterates $m - s$ times, i.e., $O(m)$ times. In each iteration of the main loop, the inner loop iterates through the Texts in a. According to Lemma 1, $n\lfloor\frac{m}{2}\rfloor$ is an upper bound to the maximum size of a Node, which implies that the inner loop does not iterate more than $n\lfloor\frac{m}{2}\rfloor$ times. In each iteration, it invokes Algorithm findMatches, whose worst-time complexity is $O(k)$, where k denotes the size of the text in which a pattern is searched [22]. This implies that $O(m)$ is an upper bound to the worst-case time complexity of the instructions inside the inner loop. As a conclusion, $O(n\lfloor\frac{m}{2}\rfloor + m\,n\lfloor\frac{m}{2}\rfloor\,m) \subseteq O(n\,m^3)$ is an upper bound to the worst-case time required to execute findPattern(a, s). □

Lemma 5 (Algorithm expand). *Let a be a Node, and s be a pattern size. $O(n\,m^3)$ is an upper bound to the worst-case time required to execute expand(a, s).*

Proof. In the worst-case, the invocation to Algorithm expand requires to invoke Algorithms findPattern and createChildren in sequence, which according to Lemmata 4 and 3 require no more than $O(n\,m^3)$ and $O(n\,m^2)$ time in the worst-case. As a conclusion, $O(n\,m^3 + n\,m^2) \subseteq O(n\,m^3)$ is an upper bound to the worst-case time required to execute expand(a, s). □

Lemma 6 (Algorithm createTrinaryTree). *Let a be a Node, min and max be the minimum and maximum sizes of the shared patterns for which the algorithm searches, respectively. $O(n\,m^5)$ is an upper bound to the worst-case time required to execute createTrinaryTree(a, min, max).*

Proof. The algorithm first iterates through every possible size between *min* and *max*, which amounts to m times in the worst-case. In each iteration, the algorithm executes Algorithm expand, which according to Lemma 5 requires no more than $O(n\,m^3)$ time. Then, $O(n\,m^4)$ is an upper bound to the first loop. If a is expanded, then another loop iterates through its leaves and executes Algorithm createTrinaryTree recursively. According to Lemma 2, $3\,m$ is an upper bound to the number of Nodes created by createTrinaryTree. As a conclusion, $O(n\,m^4\,3\,m) \subseteq O(n\,m^5)$ is an upper bound to the worst-case time required to execute createTrinaryTree(a, min, max). □

Lemma 7 (Algorithm learnTemplate). *Let a be a Node. $O(n\,m^2)$ is an upper bound to the worst-case time required to execute learnTemplate$(a, "")$.*

Proof. The algorithm works on a maximum of $3\,m$ Nodes according to Lemma 2. If a Node is a leaf, the algorithm iterates on its Texts no more than $\frac{m}{2}$ times according to Lemma 1. Otherwise, the algorithm is invoked recursively on the children of this node, namely: prefixes, separators, and suffixes. In the case of the separators, whose number is limited by the upper bound $n\lfloor\frac{m}{2}\rfloor$, the algorithm iterates on them twice to check for repeatability and searching for *nil* values. As a conclusion, $O(3\,m\,2\,n\lfloor\frac{m}{2}\rfloor) \subseteq O(n\,m^2)$ is an upper bound to the worst-case time required to execute learnTemplate$(a, "")$. □

4 Experimental Evaluation

In this section, we first describe our experimentation environment, then report on our experimental results, and finally analyse them statistically.

4.1 Experimentation Environment

We have developed a Java 1.7 prototype of our proposal using the CEDAR framework [33][1]. 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 GiB of RAM, Windows 7 Pro 64-bit, Oracle's Java Development Kit 1.7.0_02, and GNU Regex 1.1.4.

We performed our experiments on a collection 2 084 web documents from 55 datasets; 41 datasets were collected from real-world web sites and the remaining were downloaded from the RoadRunner and the RISE public repositories. The first group contains datasets on books, cars, conferences, doctors, jobs, movies, real estates, and sports. These categories were randomly sampled from The Open Directory sub-categories, and the web sites inside each category were randomly selected from the 100 best ranked web sites between December 2010 and March 2011 according to Google's search engine. We downloaded 30 web documents from each web site and handcrafted a set of annotations with the data that we would like to extract from each document. The second group contains all of

1. http://www.tdg-seville.info/Download.ashx?id=341

the datasets available at the RoadRunner repository [11] and the datasets from the RISE repository that provide semi-structured web documents [28].

The most closely-related proposals are RoadRunner [11]–[13], ExAlg [2], and FiVaTech [21]. Unfortunately, we could not find a public implementation of ExAlg or get it from the authors. To complete the collection of baseline proposals, we selected two supervised classical proposals that rank amongst the most cited in the literature, namely: SoftMealy [20] and WIEN [24]. Our goal was to prove that Trinity improves on the most closely-related proposals for which we have found an implementation and that the fact that it is unsupervised does not actually have a negative impact on its effectiveness.

4.2 Experimental Results

We ran Trinity, the latest version of RoadRunner [12], FiVaTech, SoftMealy, and WIEN on our datasets in order to learn extraction rules. Regarding Trinity, we set $min = 1$ and $max = \lfloor 0.05\,m \rfloor$, where m denotes the size in tokens of the shortest input document. We measured the standard effectiveness measures (precision, recall, and the $F1$ measure) and two efficiency measures (learning and extraction time).

In the case of SoftMealy and WIEN, it was easy to compute the precision and recall since both techniques are supervised, i.e., they require the user to provide annotations with the data to be extracted so that an extraction rule can be learnt and evaluated. Contrarily, Trinity, RoadRunner, and FiVaTech are unsupervised, i.e., they learn a rule that extracts as much data as possible, give each capturing group a computer-generated label, and it is the responsibility of the user to assign a meaning to these labels. Since we handcrafted annotations for every web document in our datasets, we could find which extracted data group was the closest to each annotation. To do so, we compared each piece of text extracted to every annotation, and computed the number of true positives (tp), false negatives (fn), and false positives (fp), since this allowed us to compute precision as $P = \frac{tp}{tp+fp}$, recall as $R = \frac{tp}{tp+fn}$, and the $F1$ measure as $F1 = 2\,\frac{P\,R}{P+R}$. Given a group of annotations, we can consider that the precision and recall to extract them corresponds to the extracted data group with the highest $F1$ measure.

Table 1 presents the results of our experiments. The columns report on the number of web documents in each dataset (N), their average size in KiB (S), average number of errors reported by JTidy (E), precision (P), recall (R), the $F1$ measure ($F1$), learning time in CPU seconds (LT), and extraction time in CPU seconds (ET); in the case of Trinity, we also report on the CPU time in seconds to learn a rule without introducing any biases (LT'), that is setting min to one and max to the size in tokens of the shortest input document. Since the results are rounded to two decimal digits, times that were smaller than 0.005 seconds are reported as 0.00 in the table, but at least four decimal digits were internally taken into account to compute the means and the standard deviations. A dash in a cell means that the corresponding technique was not able to learn an extraction rule in 15 CPU minutes or that it threw an exception. The first two rows provide a summary of these measures in terms of mean values and standard deviations.

Our first conclusion is that the bias that Trinity allows to introduce, indeed helps boost its efficiency since the average learning time is 0.13 ± 0.16 CPU seconds if the bias is introduced, whereas it is 10.82 ± 33.85 CPU seconds without the bias. Finding a good bias is always a difficult task, since it requires to fine-tune the parameters on which a proposal relies so as to find an appropriate trade-off between efficiency and effectiveness; this, in turn, requires to have a good training set on which precision and recall can be computed. We experimentally found that setting $min = 1$ and $max = \lfloor 0.05\,m \rfloor$, where m denotes the size in tokens of the shortest input document, was the maximum allowable bias, that is, a bias that boosts efficiency, but does not have a negative impact on the efficiency. In practice, our results suggest that Trinity can be used without introducing any biases since the learning times are quite competitive regarding the other proposal.

To draw more conclusions from our experimental results, we summarised the $F1$ measures, the learning times, and extraction times in Figs. 9, 10, and 11, respectively. The summaries are presented as box plots, since they help intuitively compare these measures taking into account the whole range of values.

Note that all of the techniques can achieve the maximum $F1$, which means that they can extract information with perfect precision and recall in some cases; what differentiates Trinity from the others is that its interquartile range is the smallest one and the highest one, which indicates that it can achieve high effectiveness in the majority of cases. These results suggest that the other techniques perform more irregularly and worst than Trinity regarding effectiveness.

The results are similar regarding efficiency: note that the range from the first to the third quartile is smaller for Trinity than for the other techniques regarding the learning time and similar to RoadRunner's regarding the extraction time, but clearly smaller than for the other techniques; the dispersion from the minimum values to the first quartile and from the third quartile to the maximum are also a clear indication that our technique performs more homogeneously than the others regarding learning and extraction time, with the only exception of RoadRunner regarding the extraction time. These results suggest that the other techniques perform worst than Trinity regarding efficiency, except for RoadRunner.

Table 1 also reports on the average number of errors JTidy found in each dataset. Our goal was to check that Trinity is not affected by the fact that typical web documents are malformed XHTML documents. To draw a conclusion, we plotted the $F1$ measures we gathered for every pair of datasets and techniques and the number of errors in a radial chart, cf. Fig. 12. Unfortunately, it is difficult to discern whether variations to the number of errors has an impact on the $F1$ measure. Therefore, we defer drawing a conclusion on this topic to the statistical analysis in the following section.

TABLE 1
Experimental Results

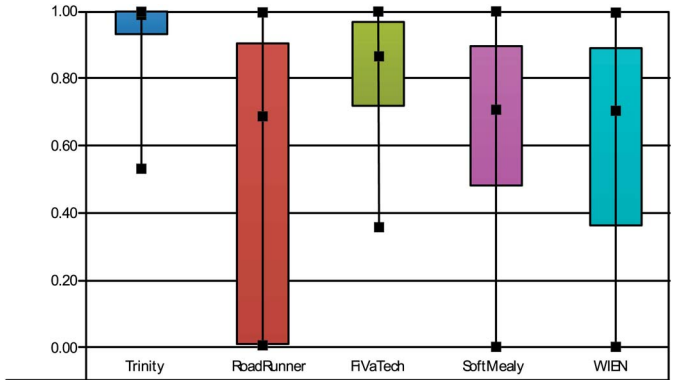
					Trinity						RoadRunner					FivaTech					SoftMealy					WIEN				
	Summary	N	S	E	P	R	F1	LT	LT'	ET	P	R	F1	LT	ET	P	R	F1	LT	ET	P	R	F1	LT	ET	P	R	F1	LT	ET
	Mean	37.89	50.90	72.08	0.96	0.95	0.95	0.13	10.82	0.02	0.57	0.55	0.55	8.80	0.01	0.79	0.88	0.82	122.94	0.24	0.84	0.61	0.66	7.10	46.30	0.72	0.61	0.64	7.80	10.93
	Standard deviation	41.67	43.56	71.80	0.07	0.11	0.09	0.16	33.85	0.03	0.38	0.40	0.39	11.11	0.01	0.21	0.16	0.17	196.42	0.36	0.15	0.32	0.30	5.13	54.45	0.24	0.31	0.29	5.15	12.88
	Dataset	N	S	E	P	R	F1	LT	LT'	ET	P	R	F1	LT	ET	P	R	F1	LT	ET	P	R	F1	LT	ET	P	R	F1	LT	ET
Books	Abe Books	30	37.65	58.73	1.00	1.00	1.00	0.03	1.00	0.00	0.65	0.58	0.61	7.19	0.01	0.92	0.99	0.95	15.46	0.12	0.87	0.58	0.70	9.09	26.96	0.52	0.16	0.24	9.66	10.26
	Awesome Books	30	20.15	43.27	1.00	0.87	0.93	0.03	0.49	0.00	0.78	0.53	0.63	3.46	0.01	0.85	1.00	0.92	8.14	0.14	1.00	0.39	0.56	5.68	16.52	0.77	0.26	0.39	5.01	6.27
	Better World Books	30	125.23	46.00	0.99	1.00	0.99	0.17	16.14	0.00	-	-	-	12.59	-	0.99	0.96	0.97	85.32	0.39	0.98	0.99	0.98	16.15	41.39	0.43	0.35	0.39	15.57	17.04
	Many Books	30	26.84	130.00	0.99	0.99	0.99	0.11	0.74	0.02	0.97	0.95	0.96	1.87	0.01	0.77	0.97	0.86	65.49	0.12	0.99	0.99	0.99	7.38	30.33	0.25	0.23	0.24	6.74	8.14
	Waterstones	30	79.68	129.10	0.96	1.00	0.98	0.07	1.83	0.00	0.98	0.87	0.92	6.18	0.01	1.00	0.94	0.97	51.53	1.98	1.00	1.00	1.00	8.67	77.36	0.71	0.67	0.69	8.02	8.72
Movies	IMDB	30	97.35	12.00	0.93	0.86	0.89	0.45	0.80	0.03	0.39	0.35	0.37	18.39	0.02	-	-	-	-	-	0.88	0.85	0.86	16.38	63.24	0.38	0.38	0.38	15.87	16.46
	Disney Movies	30	47.26	59.40	1.00	1.00	1.00	0.07	6.37	0.00	0.67	0.67	0.67	3.03	0.01	0.71	0.67	0.69	259.23	0.08	0.97	0.97	0.97	4.34	44.69	0.72	0.72	0.72	3.95	3.82
	Albania Movies	30	5.70	20.90	0.95	0.98	0.96	0.02	0.20	0.00	0.75	0.77	0.76	1.45	0.00	0.82	0.81	0.81	1.59	0.00	0.85	0.40	0.54	3.65	3.45	0.87	0.24	0.38	2.70	1.67
	All Movies	30	33.79	32.33	0.97	0.96	0.96	0.21	0.67	0.00	0.27	0.26	0.26	4.32	0.01	0.79	0.74	0.77	14.84	0.11	0.93	0.29	0.44	11.29	38.86	0.13	0.07	0.09	7.78	5.91
	CITWF	30	19.50	138.80	1.00	1.00	1.00	0.02	40.75	0.00	0.94	0.94	0.94	1.40	0.00	1.00	1.00	1.00	29.70	0.05	0.92	0.72	0.81	3.70	10.39	0.39	0.30	0.34	2.79	3.79
	Soul Films	30	28.48	66.13	0.99	0.92	0.95	0.03	0.66	0.00	0.50	0.50	0.50	4.54	0.01	0.59	1.00	0.74	17.24	0.05	0.99	0.95	0.97	10.64	37.58	0.91	0.81	0.86	9.73	9.36
Cars	Auto Trader	30	183.51	273.23	0.99	1.00	1.00	0.28	32.46	0.03	-	-	-	-	-	-	-	-	-	-	0.89	0.87	0.88	14.87	103.82	0.89	0.00	0.00	14.18	14.02
	Car Max	30	67.26	191.47	1.00	1.00	1.00	0.21	10.23	0.02	0.98	0.98	0.98	12.07	0.01	0.45	0.89	0.60	34.21	0.20	0.89	0.89	0.89	7.66	22.90	0.88	0.88	0.88	7.43	7.63
	Car Zone	30	71.05	118.80	0.98	1.00	0.99	0.04	1.85	0.00	0.72	0.81	0.76	4.88	0.03	0.92	1.00	0.96	446.90	0.59	0.92	0.02	0.05	5.30	28.88	0.82	0.83	0.83	4.80	4.76
	Classic Cars for Sale	30	76.02	25.03	0.86	0.90	0.88	0.11	3.12	0.02	0.49	0.49	0.49	51.14	0.01	-	-	-	-	-	0.90	0.90	0.90	10.92	78.55	0.17	0.13	0.15	10.03	10.28
	Internet Autoguide	30	154.22	163.90	1.00	1.00	1.00	0.12	11.45	0.05	0.92	1.00	0.95	4.02	0.01	0.97	0.94	0.96	117.16	0.19	-	-	-	8.74	-	0.11	0.11	0.11	7.78	9.02
Events	Linked In	30	9.89	23.67	0.96	0.96	0.96	0.04	0.09	0.02	0.53	0.53	0.53	1.37	0.00	-	-	-	-	-	0.87	0.55	0.68	5.04	42.04	0.57	0.20	0.30	4.15	3.34
	All Conferences	30	17.83	30.47	0.98	0.99	0.99	0.11	1.95	0.00	0.72	0.72	0.72	2.90	0.00	0.84	0.90	0.87	42.96	0.08	0.99	0.25	0.40	10.06	62.24	0.80	0.40	0.53	7.27	4.74
	Mbendi	30	6.95	27.00	1.00	1.00	1.00	0.01	0.12	0.00	0.87	0.87	0.87	0.72	0.00	0.90	1.00	0.95	1.56	0.03	0.60	0.60	0.60	2.61	5.74	0.80	0.40	0.53	1.86	1.28
	Net Lib	30	2.13	7.00	0.96	0.98	0.97	0.00	0.00	0.00	0.00	0.00	0.00	0.12	0.00	0.39	0.50	0.44	0.27	0.03	0.87	0.44	0.59	6.65	7.86	0.40	0.40	0.40	4.43	2.31
	RD Learning	30	4.23	14.00	0.99	0.99	0.99	0.02	0.02	0.00	0.77	0.77	0.77	0.51	0.00	0.99	0.79	0.88	6.21	0.02	0.52	0.39	0.45	4.62	7.10	0.62	0.23	0.34	3.24	1.95
Doctors	Web MD	30	59.23	24.10	1.00	1.00	1.00	0.02	2.54	0.00	0.06	0.06	0.06	25.26	0.01	0.77	1.00	0.87	11.81	0.03	0.86	0.45	0.59	8.28	29.94	0.60	0.60	0.60	9.45	14.54
	Ame. Medical Assoc.	30	24.87	36.00	0.98	1.00	0.99	0.02	0.15	0.02	-	-	-	5.57	-	-	-	-	-	-	0.79	0.39	0.53	6.07	10.53	0.60	0.60	0.60	6.99	7.38
	Dentists	30	11.92	103.27	0.92	1.00	0.96	0.01	0.15	0.00	0.96	0.96	0.96	0.89	0.00	0.56	0.99	0.72	9.94	0.08	0.61	0.61	0.61	2.28	8.16	1.00	1.00	1.00	1.97	1.84
	Dr. Score	30	23.78	33.07	1.00	1.00	1.00	0.03	0.33	0.00	0.84	1.00	0.91	1.47	0.00	0.78	1.00	0.88	25.35	0.06	0.91	0.87	0.89	4.49	17.50	0.78	0.80	0.79	3.88	3.62
	Steady Health	30	81.39	24.00	1.00	1.00	1.00	0.67	17.87	0.02	1.00	1.00	1.00	11.33	0.03	0.83	0.83	0.83	9.59	0.11	0.75	0.25	0.38	9.70	40.00	0.75	0.75	0.75	9.56	9.77
Jobs	Insight into Diversity	30	30.36	67.07	0.83	0.83	0.83	0.04	0.39	0.00	0.57	0.57	0.57	4.32	0.01	1.00	0.74	0.85	13.76	0.11	0.57	0.47	0.52	7.77	16.72	1.00	1.00	1.00	7.49	7.14
	4 Jobs	30	79.76	110.47	0.92	0.98	0.95	0.09	3.07	0.02	0.00	0.00	0.00	2.98	0.01	-	-	-	-	-	0.45	0.25	0.32	9.09	36.60	0.94	0.94	0.94	9.02	9.50
	6 Figure Jobs	30	72.79	169.37	1.00	1.00	1.00	0.04	7.40	0.02	0.52	0.52	0.52	21.26	0.02	1.00	0.98	0.99	90.78	0.34	0.75	0.00	0.00	11.76	26.79	0.25	0.25	0.25	11.56	11.93
	Career Builder	30	54.17	93.97	1.00	1.00	1.00	0.05	1.29	0.00	0.00	0.00	0.00	5.34	0.01	0.80	0.83	0.82	266.00	0.31	0.75	0.00	0.00	8.13	50.72	0.75	0.75	0.75	8.11	7.72
	Job of Mine	30	23.90	41.03	0.86	1.00	0.93	0.06	0.78	0.00	0.72	0.72	0.72	2.18	0.01	-	-	-	-	-	0.75	0.03	0.06	5.19	22.56	0.50	0.50	0.50	5.09	4.77
Real Estate	Yahoo!	30	93.94	292.27	1.00	1.00	1.00	0.15	24.44	0.02	-	-	-	15.66	-	0.77	0.97	0.86	246.95	0.89	1.00	1.00	1.00	16.32	86.55	0.83	0.83	0.83	15.62	16.75
	Haart	30	89.64	40.00	1.00	1.00	1.00	0.05	2.02	0.00	0.79	0.79	0.79	6.97	0.01	0.94	1.00	0.97	20.76	0.09	1.00	1.00	1.00	7.43	100.85	0.78	0.75	0.76	7.04	6.96
	Homes	30	59.32	99.93	0.99	0.99	0.99	0.10	2.12	0.00	1.00	1.00	1.00	3.96	0.01	-	-	-	-	-	0.80	0.79	0.79	8.25	65.88	0.92	0.92	0.92	8.22	8.17
	Remax	30	69.98	75.70	0.70	0.98	0.82	0.20	38.33	0.02	0.00	0.00	0.00	14.94	0.01	-	-	-	-	-	0.84	0.84	0.84	7.60	25.65	1.00	1.00	1.00	7.52	7.49
	Trulia	30	175.39	312.73	0.63	1.00	0.77	0.79	240.03	0.05	0.00	0.00	0.00	46.78	0.03	-	-	-	-	-	0.88	0.92	0.90	25.55	284.94	1.00	0.89	0.94	25.18	25.37
Sports	Player Profiles	30	20.89	35.07	1.00	1.00	1.00	0.11	0.91	0.23	-	-	-	-	-	0.36	0.99	0.52	14.68	0.06	0.83	0.13	0.23	5.51	17.43	1.00	1.00	1.00	5.34	4.96
	UEFA	30	63.42	31.80	1.00	1.00	1.00	0.05	1.56	0.00	1.00	1.00	1.00	6.15	0.03	-	-	-	-	-	1.00	1.00	1.00	5.99	33.23	1.00	1.00	1.00	5.83	5.71
	ATP World Tour	30	135.55	92.03	0.97	0.99	0.98	0.29	61.21	0.03	0.97	0.97	0.97	16.57	0.04	0.99	0.88	0.93	111.03	0.83	0.94	0.94	0.94	19.14	182.38	0.71	0.71	0.71	18.75	18.33
	NFL	30	94.92	84.16	1.00	1.00	1.00	0.19	2.68	0.00	0.98	0.98	0.98	34.30	0.03	0.53	0.81	0.64	159.39	0.39	0.71	0.71	0.71	10.14	39.91	0.86	0.86	0.86	9.64	9.56
	Soccer Base	30	85.02	156.37	0.97	1.00	0.98	0.27	32.18	0.03	0.95	1.00	0.97	14.93	0.01	-	-	-	-	-	0.89	0.89	0.89	11.54	47.80	0.89	0.89	0.89	11.04	10.87
RoadRunner	Amazon (cars)	21	25.16	20.00	0.93	0.73	0.82	0.01	0.05	0.00	0.00	0.00	0.00	1.36	0.01	0.60	0.67	0.63	2.29	0.05	0.98	1.00	0.99	1.06	10.09	0.97	1.00	0.98	19.47	8.10
	UEFA (players)	20	12.09	10.40	1.00	0.90	0.95	0.02	0.15	0.02	1.00	1.00	1.00	0.62	0.00	0.91	0.94	0.92	10.81	0.05	0.92	0.90	0.91	1.34	11.64	0.92	0.51	0.66	3.53	3.70
	Amazon (popartist)	19	34.17	35.00	1.00	0.98	0.99	0.02	0.17	0.00	0.00	0.00	0.00	2.45	0.00	1.00	1.00	1.00	246.97	0.11	0.94	0.72	0.82	3.42	23.35	0.92	0.58	0.71	15.66	10.22
	UEFA (teams)	20	6.87	32.80	0.99	0.99	0.99	0.03	0.18	0.00	0.92	0.92	0.92	0.45	0.00	0.97	0.99	0.98	0.89	0.02	0.81	0.89	0.85	1.00	3.32	0.64	0.75	0.69	1.79	2.28
	Aus Open	29	41.22	66.73	1.00	1.00	1.00	0.26	9.57	0.02	0.70	0.85	0.77	4.84	0.01	0.24	0.82	0.37	132.24	0.27	0.65	0.28	0.39	1.98	22.06	0.67	0.32	0.43	9.67	16.96
	E-Bay	50	26.43	18.12	0.97	1.00	0.98	0.51	5.56	0.08	0.82	0.82	0.82	3.73	0.01	0.83	1.00	0.91	577.97	0.48	0.70	0.12	0.20	2.23	16.04	0.70	0.12	0.20	5.44	19.55
	Major League	9	40.10	26.00	0.98	0.55	0.70	0.04	4.17	0.00	0.00	0.00	0.00	3.28	0.01	0.99	1.00	0.99	158.33	0.06	0.99	0.46	0.63	1.29	19.61	0.65	0.33	0.44	3.60	6.13
	Netflix	50	43.90	125.86	0.99	0.99	0.99	0.18	3.11	0.02	0.76	0.79	0.78	11.62	0.01	0.82	0.80	0.81	706.64	0.76	0.94	0.82	0.88	3.68	34.15	0.99	0.99	0.99	7.47	31.54
	RPM Find	20	34.68	9.90	0.95	0.97	0.96	0.03	0.05	0.05	0.10	0.00	0.00	1.28	0.01	-	-	-	-	-	0.72	0.03	0.06	1.40	26.22	0.99	0.99	0.99	1.29	10.42
RISE	Bigbook	235	24.73	20.61	0.95	0.94	0.94	0.12	0.14	0.05	0.01	0.00	0.00	25.71	0.00	-	-	-	-	-	0.81	0.77	0.79	1.50	166.23	0.68	0.98	0.80	2.68	89.14
	IAF	252	14.24	13.20	0.84	0.38	0.52	0.12	0.19	0.00	0.90	0.09	0.17	0.70	0.00	0.53	0.69	0.60	14.41	0.06	0.37	0.43	0.40	1.39	13.34	1.00	1.00	1.00	2.00	2.40
	Okra	10	7.76	15.42	1.00	0.82	0.90	0.08	0.50	0.00	0.12	0.01	0.02	20.51	0.00	0.49	0.34	0.40	849.27	0.37	0.83	0.82	0.82	1.73	216.36	0.60	0.67	0.63	1.64	35.08
	LA Weekly	28	5.16	4.93	0.97	0.92	0.94	0.01	0.01	0.00	0.00	0.00	0.00	0.47	0.00	0.83	0.57	0.68	4.24	0.05	0.87	0.49	0.63	2.18	23.45	0.90	0.80	0.85	0.90	2.73
	Zagat	91	18.23	31.92	1.00	0.86	0.92	0.17	1.02	0.05	0.00	0.00	0.00	6.60	0.00	1.00	0.98	0.99	158.48	0.06	0.81	0.63	0.71	2.57	20.39	0.81	0.72	0.76	13.60	19.75

4.3 Statistical Analysis

To confirm that the conclusions we have drawn from our empirical evaluation are valid, we need to perform a statistical analysis [31]. This consists in performing a statistical ranking regarding our performance measures and determining if there is a significant correlation from the number of errors to the effectiveness of the techniques we have evaluated.

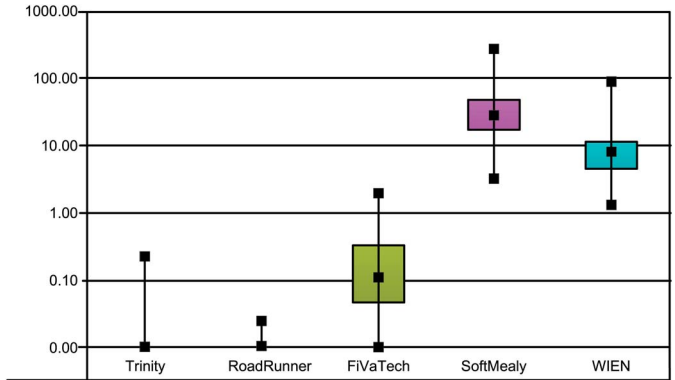
We have conducted a Shapiro-Wilk test at the standard significance level $\alpha = 0.05$ on every measure and we have found out that none of them behaves normally. As a conclusion, we have used non-parametric analysis techniques. The steps were the following: a) compute the rank of each technique from the evaluation results; b) determine if the differences in ranks are significant or not using Iman-Davenport's test; c) if the differences are significant, then compute the statistical ranking using Bergmann-Hommel's test on every pair of techniques.

Table 2 presents the results of the analysis. (Due to space limitations, we only report on the adjusted p-values, not on the exact values of the statistics.) Note that the p-value of Iman-Davenport's statistic is nearly zero in every case, which is a strong indication that there are statistically significant differences in the ranks we have computed from



	Trinity	RoadRunner	FiVaTech	SoftMealy	WIEN
Minimum	0.52	0.00	0.37	0.00	0.00
Quartile 1	0.95	0.03	0.72	0.47	0.39
Median	0.99	0.69	0.87	0.71	0.71
Quartile 3	1.00	0.92	0.96	0.89	0.87
Maximum	1.00	1.00	1.00	1.00	1.00

Fig. 9. Comparison of $F1$ measures.



	Trinity	RoadRunner	FiVaTech	SoftMealy	WIEN
Minimum	0.00	0.00	0.00	3.32	1.28
Quartile 1	0.00	0.00	0.05	16.90	4.75
Median	0.00	0.01	0.11	27.92	8.10
Quartile 3	0.02	0.01	0.31	47.02	11.40
Maximum	0.23	0.04	1.98	284.94	89.14

Fig. 11. Comparison of extraction times.

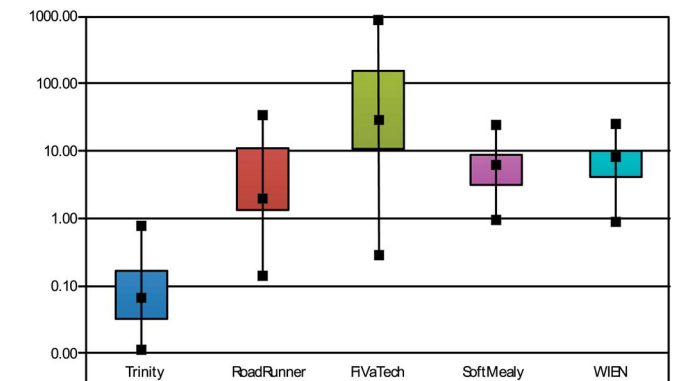
our experiments. It then proceeds to rank the techniques pairwise using Bergmann-Hommel's test. For the sake of readability, we also provide an explicit ranking in the last column. Note that our proposal ranks the first regarding every effectiveness and efficiency measures; the only tie is regarding extraction time, in which case the difference with respect to RoadRunner does not seem to be statistically significant. As a conclusion, our experiments prove that there is enough statistical evidence to conclude that our proposal outperform the others.

Recall from the previous section that we could not draw an intuitive conclusion on whether the errors in the input web documents have an impact on the effectiveness of the techniques we have compared. To discern if there is such an impact from an statistical point of view, we need to calculate the correlation from the number of errors to the $F1$ measure using Kendall's τ procedure. Table 3 presents the results of this procedure. Note that the p-value of the correlation coefficients is smaller than the standard significance level $\alpha = 0.05$ except for the case of RoadRunner and FiVaTech; in these cases the correlation coefficient is negative, which means that the effectiveness of these techniques is expected to decrease as the number of errors in the input documents increases. As a conclusion, our experiments do not show any statistical evidence that the effectiveness of our technique is sensitive to malformed input documents.

5 Related Work

In the introduction, we listed and classified many of the proposals on data extraction that we have found in the literature. Our conclusion was that Trinity is closely related to RoadRunner [11]–[13], ExAlg [2], and FiVaTech [21], which are the other three proposals that learn a regular expression that models the template used to generate the input documents. This is the reason why we restrict our conceptual comparison to them.

RoadRunner was originally proposed by [13]. It works on a collection of web documents and compares them side by side in order to infer a union-free regular expression that describes their template. Although the proposal works on the text of the input documents, it requires them to be repaired beforehand using tools like JTidy since the algorithm needs the input documents to be well-formed. The proposal constructs the extraction rule incrementally by means of a string alignment algorithm that is specifically tailored to XHTML. The initial rule is set to any of the input documents, and it is then used to parse the others. During parsing, the algorithm may find mismatches



	Trinity	RoadRunner	FiVaTech	SoftMealy	WIEN
Minimum	0.00	0.12	0.27	1.00	0.90
Quartile 1	0.03	1.47	10.81	3.01	3.92
Median	0.07	4.32	29.70	6.07	7.43
Quartile 3	0.17	12.07	158.33	9.40	9.66
Maximum	0.79	51.14	849.27	25.55	25.18

Fig. 10. Comparison of learning times.

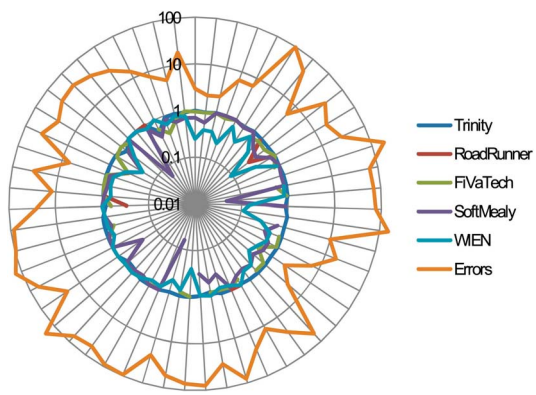


Fig. 12. Correlation from number of errors to the $F1$ measure.

TABLE 2
Results of the Statistical Ranking

Measure	Sample ranking		Iman-Davenport's test	Bergmann-Hommels's test						Statistical ranking	
	Technique	Rank	P-Value	P-Value	Trinity	RoadRunner	FiVaTech	SoftMealy	WIEN	Technique	Rank
P	Trinity	1.63	2.64E-15	Trinity	-	1.20E-12	2.23E-06	1.27E-04	3.82E-09	Trinity	1
	SoftMealy	2.88		RoadRunner		-	4.76E-02	6.77E-03	4.33E-01	SoftMealy	2
	FiVaTech	3.14		FiVaTech			-	4.33E-01	4.33E-01	FiVaTech	2
	WIEN	3.49		SoftMealy				-	1.30E-01	WIEN	2
	RoadRunner	3.86		WIEN					-	RoadRunner	2
R	Technique	Rank	P-Value	P-Value	Trinity	RoadRunner	FiVaTech	SoftMealy	WIEN	Technique	Rank
	Trinity	1.57	1.25E-18	Trinity	-	1.72E-11	4.06E-03	2.95E-10	5.69E-11	Trinity	1
	FiVaTech	2.56		RoadRunner		-	9.84E-04	1.00E+00	1.00E+00	FiVaTech	2
	SoftMealy	3.54		FiVaTech			-	4.06E-03	1.26E-03	SoftMealy	3
	WIEN	3.63		SoftMealy				-	1.00E+00	WIEN	3
	RoadRunner	3.70		WIEN					-	RoadRunner	3
F1	Technique	Rank	P-Value	P-Value	Trinity	RoadRunner	FiVaTech	SoftMealy	WIEN	Technique	Rank
	Trinity	1.48	1.41E-17	Trinity	-	2.36E-12	1.02E-05	2.10E-09	2.44E-11	Trinity	1
	FiVaTech	2.90		RoadRunner		-	5.23E-02	7.94E-01	7.94E-01	FiVaTech	2
	SoftMealy	3.35		FiVaTech			-	2.63E-01	7.70E-02	SoftMealy	2
	WIEN	3.57		SoftMealy				-	7.94E-01	WIEN	2
	RoadRunner	3.69		WIEN					-	RoadRunner	2
LT	Technique	Rank	P-Value	P-Value	Trinity	RoadRunner	FiVaTech	SoftMealy	WIEN	Technique	Rank
	Trinity	1.00	1.53E-63	Trinity	-	2.88E-08	4.35E-35	1.43E-15	5.76E-11	Trinity	1
	RoadRunner	2.71		RoadRunner		-	3.73E-11	3.40E-02	2.78E-01	RoadRunner	2
	WIEN	3.04		FiVaTech			-	2.83E-05	2.83E-08	WIEN	3
	SoftMealy	3.47		SoftMealy				-	1.48E-01	SoftMealy	3
	FiVaTech	4.78		WIEN					-	FiVaTech	4
ET	Technique	Rank	P-Value	P-Value	Trinity	RoadRunner	FiVaTech	SoftMealy	WIEN	Technique	Rank
	RoadRunner	1.53	1.71E-127	Trinity	-	9.04E-01	4.83E-06	1.83E-29	5.56E-16	Trinity	1
	Trinity	1.50		RoadRunner		-	4.83E-06	7.57E-30	4.03E-16	RoadRunner	1
	FiVaTech	2.96		FiVaTech			-	8.71E-11	9.39E-04	FiVaTech	2
	WIEN	4.02		SoftMealy				-	2.79E-03	WIEN	3
	SoftMealy	4.98		WIEN					-	SoftMealy	4

between the partial rule and the current input document, in which case a number of generalisation strategies are used; simply put, these strategies try to find out if a new repetitive or optional structure needs to be included in the partial rule so that it can satisfactorily parse the current input document. The process continues until every input document has been parsed and used to generalise the partial rule thus constructed. The time complexity of the algorithm was proven to be exponential in the number of tokens of the input documents; the authors introduced several biases to the generalisation strategies in order to lower the time complexity, namely: limiting the number of alternatives to be explored, the number of backtracks to be performed, and discarding some regular sub-expressions. The algorithm was proven to perform well in practice thanks to the previous biases, but no formal proof regarding its resulting time or space complexity was presented. Unfortunately, the biases had a negative impact on its effectivity. Later, [11] presented a new version of the algorithm that was proven to be polynomial for a subclass of union-free regular expressions that is called prefix markup. Unfortunately, according to the experiments that the authors carried out, roughly 50% of the sites they analysed were not prefix markup. This motivated them to work on a technique to transform a regular web document into another that is prefix mark-up [12]. This technique is applied as a pre-processing step to RoadRunner and proved to boost its effectiveness. The technique is exponential because it includes a module to perform disambiguation that is an instance of the set partitioning problem, which is known to be NP-complete. The authors designed a number of heuristics that help reduce its complexity in many common cases.

TABLE 3
Results of the Statistical Analysis of Correlation

	Correlation coefficient	P-Value
Trinity	0.01	9.32E-01
RoadRunner	-0.36	7.00E-03
FiVaTech	-0.27	4.70E-02
SoftMealy	0.04	7.58E-01
WIEN	0.00	9.90E-01

Trinity is also based on aligning the input documents to try to discover their common template, but it differs significantly from RoadRunner, namely: a) Both proposals, work on the text of the input documents, but RoadRunner requires them to be well-formed, whereas Trinity does not; we experimentally found that there is a statistically significant and negative correlation from the number of errors in the input documents to the effectiveness of RoadRunner, which is not the case of Trinity. b) Trinity aligns all of the input documents in parallel, whereas RoadRunner aligns a partial rule to a unique document to produce a new version of the rule. c) Trinity searches for shared patterns and infers repetitions and optionals from the texts and separators in a trinary node, whereas RoadRunner searches for mismatches and then tries to find out if they must be generalised to a capturing group, a repetition, or an optional expression, which is a complex procedure that requires backtracking and has many special cases. d) Trinity is polynomial in both space and time, whereas RoadRunner was proven to be polynomial in time for a subset of union-free regular expressions, but remains exponential for general union-free regular expressions; unfortunately, no results about RoadRunner's space complexity are available in the literature. e) Trinity relies on two parameters that help introduce a bias to its algorithm to search for shared patterns; note that it is not mandatory to introduce this bias for the algorithm to work well, and that introducing it helps increase its efficiency without sacrificing its effectiveness; contrarily RoadRunner requires a bias to be introduced so that it is efficient enough for practical purposes; unfortunately, this bias has a negative impact on its effectiveness; the version by [11] was polynomial but it requires the input documents to be generated by a prefix mark-up template, which is not the most common kind of template. Introducing a preprocessing step to transform the input document into prefix mark-up equivalents also has a significant impact on complexity.

ExAlg was proposed by [2]. It works in two stages: it first computes so-called large and frequently occurring equivalence classes of tokens, or LFEQs for short, and then learns a regular expression and a data schema from them. An LFEQ is a sufficiently large maximal subset of tokens that occur a sufficiently large and equal number of times in every input document. Using a simple padding technique, the authors can guarantee that there always exist a unique LFEQ in which the occurrence frequency of each token is exactly one; such an LFEQ is referred to as the root LFEQ. Note that computing a set of LFEQs from an input set of documents is relatively simple; the complex part of the proposal is to purge them to exclude invalid LFEQs, which are LFEQs whose tokens do not always appear in the same relative order or LFEQs that are not nested within other LFEQs, that is, whose tokens do not always occur in the same context within other LFEQs. Furthermore, a token may have different roles in the same document, e.g., the role of a token like Author is not the same when it occurs in the middle of a paragraph and when it occurs in <td>Author:</td>; ExAlg considers that two tokens with the same lexeme are different if they have different paths in the parse tree or if they appear in different contexts, that is, the surrounding tokens are different in different occurrences. The authors sketched an algorithm to find an initial set of LFEQs and refine them by discarding invalid ones and differentiating roles. The resulting LFEQs are then searched recursively for regular patterns; unfortunately, the algorithm to find such patterns was just sketched. The regular expression that is computed for the root LFEQ is the one that models the template used to generate the input document. Unfortunately, we could not find any written record regarding the complexity of ExAlg. The authors mentioned that the first stage run linearly in their experiments, but no formal proof was presented. Neither is it easy to infer the complexity because the authors did not provide an algorithm, but some definitions and an intuitive sketch of how their proposal works. The authors, however, made it explicit the assumptions that must hold for their proposal to work well, namely: a) a large number of template tokens must have unique roles; b) a large number of tokens must be associated with each type constructor (i.e., capturing group, union, repetition, or optionality) and each type constructor must be instantiated a large number of times in each input document; c) there must not be any regularities in the data that can lead to fake LFEQs; and d) there must be separators around data.

Trinity also builds on the idea that tokens that are shared amongst a number of documents are likely to belong to the template used to generate them, but our technique differs significantly from ExAlg, namely: a) It is not clear whether ExAlg can work on malformed input documents or not; apparently, the core of the algorithm works on strings of tokens, but it requires to compute their paths in the corresponding parse trees to differentiate their roles. To create a parse tree, the input web documents need to be repaired if they are not well-formed, which we have found has a negative impact on the effectiveness of RoadRunner and FiVaTech; contrarily, Trinity works on the input documents as they are provided b) To some extent, ExAlg creates kind of a tree when it searches for the LFEQs that are nested into another LFEQ; note that, to some extent, each node in a trinary tree might be considered kind of a LFEQ, the difference being that a trinary node focuses on the longest patterns found in the parent node and that they are inherently nested and aligned with respect to such parent nodes; in other words: no trinary node may be invalid in the sense of ExAlg, which makes it totally unnecessary to refine the trinary nodes our algorithm creates. c) ExAlg can learn disjunctions, but, unfortunately, the algorithm to learn them was not detailed; Trinity cannot learn disjunctions, but we did not find that a serious shortcoming since the majority of cases in which there was a disjunction could be dealt with by just considering that formatting tags like strong, emph or strike are text. d) ExAlg builds on four assumptions and the authors found that some of them did not hold in some experimental datasets; chiefly, the second assumption may prevent ExAlg from working well with documents whose structure is very simple, whereas Trinity builds on only one assumption: that the input documents were generated by the same server-side template. e) Trinity was proven to be polynomial in both time and space, whereas no formal proof on ExAlg's complexity was found in the literature. f) ExAlg relies on two parameters to determine when a prospective LFEQ is large enough and its tokens are recurring enough to be considered a proper LFEQ; both

parameters introduce a bias that has an impact on the effectiveness of the proposal; contrarily, Trinity does not require any biases to be introduced unless we need to boost its efficiency: note that our experimental results confirm that even without introducing the biases, Trinity is very efficient for practical purposes. g) [21] highlighted that ExAlg does not try to align the input documents and that the token differentiation criterion does not take into account the subtree below tag tokens; they found out that these issues lead to accidental LFEQs that misalign the first and the last records of a list and that the resulting regular expressions include many disjunctive and empty expressions that do not extract any data. None of the previous issues applies to Trinity.

FiVaTech was proposed by [21]. It proceeds in two stages, namely: the first stage decomposes the input documents into a collection of DOM trees that are then merged into a so-called pattern tree from which it is relatively simple to infer a regular expression that models the template used to generate the input documents; the second stage cleans the pattern tree to produce a scheme of the data that the regular expression extracts. In the first stage, the DOM trees of the input documents are compared level by level building on a two-tree matching algorithm that returns a similarity score; nodes whose similarity score are above a user-defined parameter are assigned the same symbol and considered peer nodes. Peer nodes are then aligned using a matrix alignment algorithm; the resulting matrix is then mined for the longest repetitive patterns and then for optional nodes; optional nodes that are adjacent and have complementary occurrence vectors are considered disjunctions. The resulting pattern tree represents a regular expression in which leaf nodes are capturing groups, some intermediate nodes represent the tags of the template, and some others the repetition or the optional regular operators. The authors mentioned the time complexity of some of the sub-algorithms on which their proposal relies, but no formal proof regarding the overall time and space complexity was presented.

The approach that we used to devise Trinity is quite different, namely: a) FiVaTech relies on DOM trees. This requires to parse the input documents and correct them, which we have found experimentally has a negative impact on its effectiveness; contrarily, Trinity can work on malformed input documents without correcting them. b) FiVaTech also searches for the longest repeating patterns, but this is done after peer nodes are identified and their children aligned, which is a process that requires a time that is not negligible; contrarily, Trinity first identifies the longest shared patterns and then determines which nodes represent repetitions or optionality using quite a simple criterion whose computational cost is negligible. c) FiVaTech can detect repetition patterns only regarding the children of a node, whereas Trinity does not impose this limitation. d) FiVaTech relies on a parameter that biases the procedure to determine whether two nodes are peer nodes or not; the selection of a proper value is not easy and has an impact on the effectiveness; contrarily, the biases in Trinity are not mandatory and can be used for efficiency purposes only. e) We have proved that Trinity is polynomial in both time and space; unfortunately, no record about FiVaTech's overall complexity was found in the literature.

6 Conclusion

We have presented an effective and efficient unsupervised data extractor called Trinity. It is based on the hypothesis that web documents generated by the same server-side template share patterns that do not provide any relevant data, but help delimit them. The rule learning algorithm searches for these patterns and creates a trinary tree, which is then used to learn a regular expression that represents the template that was used to generate input web documents. Our experiments on 55 real-world web sites proved that our technique achieves very high precision and recall with a learning and extraction time that is almost negligible. Furthermore, errors in the input XHTML documents do not have a negative impact on its effectiveness. We also identified a means to introduce a bias to the search procedure that improves its efficiency without a negative impact on its effectiveness.

Acknowledgments

The authors thank the referees for their insightful comments on earlier versions of this article and the authors of the related proposals for sharing their implementations with us. Our results were supported by local R&D programmes and FEDER funds through grants 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.

References

[1] M. Álvarez, A. Pan, J. Raposo, F. Bellas, and F. Cacheda, "Extracting lists of data records from semi-structured web pages," *Data Knowl. Eng.*, vol. 64, no. 2, pp. 491–509, Feb. 2008.

[2] A. Arasu and H. Garcia-Molina, "Extracting structured data from web pages," in *Proc. 2003 ACM SIGMOD*, San Diego, CA, USA, pp. 337–348.

[3] J. L. Arjona, R. Corchuelo, D. Ruiz, and M. Toro, "From wrapping to knowledge," *IEEE Trans. Knowl. Data Eng.*, vol. 19, no. 2, pp. 310–323, Feb. 2007.

[4] F. Ashraf, T. Özyer, and R. Alhajj, "Employing clustering techniques for automatic information extraction from HTML documents," *IEEE Trans. Syst. Man Cybern. C*, vol. 38, no. 5, pp. 660–673, Sept. 2008.

[5] M. E. Califf and R. J. Mooney, "Bottom-up relational learning of pattern matching rules for information extraction," *J. Mach. Learn. Res.*, vol. 4, pp. 177–210, May 2003.

[6] A. Carlson and C. Schafer, "Bootstrapping information extraction from semi-structured web pages," in *Proc. ECML/PKDD*, Berlin, Germany, 2008, pp. 195–210.

[7] C.-H. Chang and S.-C. Kuo, "OLERA: Semisupervised web-data extraction with visual support," *IEEE Intell. Syst.*, vol. 19, no. 6, pp. 56–64, Nov./Dec. 2004.

[8] C.-H. Chang and S.-C. Lui, "IEPAD: Information extraction based on pattern discovery," in *Proc. 10th Int. Conf. WWW*, Hong Kong, China, 2001, pp. 681–688.

[9] C.-H. Chang, M. Kayed, M. R. Girgis, and K. F. Shaalan, "A survey of web information extraction systems," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 10, pp. 1411–1428, Oct. 2006.

[10] W. W. Cohen, M. Hurst, and L. S. Jensen, "A flexible learning system for wrapping tables and lists in HTML documents," in *Proc. 11th Int. Conf. WWW*, 2002, pp. 232–241.

[11] V. Crescenzi and G. Mecca, "Automatic information extraction from large websites," *J. ACM*, vol. 51, no. 5, pp. 731–779, Sept. 2004.

[12] V. Crescenzi and P. Merialdo, "Wrapper inference for ambiguous web pages," *Appl. Artif. Intel.*, vol. 22, no. 1–2, pp. 21–52, Jan. 2008.

[13] V. Crescenzi, G. Mecca, and P. Merialdo, "Road runner: Towards automatic data extraction from large web sites," in *Proc. 27th Int. Conf. VLDB*, Rome, Italy, 2001, pp. 109–118.

[14] H. Elmeleegy, J. Madhavan, and A. Y. Halevy, "Harvesting relational tables from lists on the web," in *Proc. VLDB*, vol. 2, no. 1, pp. 1078–1089, Aug. 2009.
[15] D. Freitag, "Information extraction from HTML: Application of a general machine learning approach," in *Proc. 15th Nat/10th Conf. AAAI/IAAI*, Menlo Park, CA, USA, 1998, pp. 517–523.
[16] P. Gulhane, R. Rastogi, S. H. Sengamedu, and A. Tengli, "Exploiting content redundancy for web information extraction," in *Proc. 19th Int. Conf. WWW*, Raleigh, NC, USA, 2010, pp. 1105–1106.
[17] P. Gulhane *et al.*, "Web-scale information extraction with vertex," in *IEEE 27 ICDE*, Hannover, Germany, 2011, pp. 1209–1220.
[18] R. Gupta and S. Sarawagi, "Answering table augmentation queries from unstructured lists on the web," in *Proc. VLDB*, vol. 2, no. 1, pp. 289–300, Aug. 2009.
[19] J. L. Hong, E.-G. Siew, and S. Egerton, "Information extraction for search engines using fast heuristic techniques," *Data Knowl. Eng.*, vol. 69, no. 2, pp. 169–196, Feb. 2010.
[20] C.-N. Hsu and M.-T. Dung, "Generating finite-state transducers for semi-structured data extraction from the web," *Inform. Syst.*, vol. 23, no. 8, pp. 521–538, Dec. 1998.
[21] M. Kayed and C.-H. Chang, "FiVaTech: Page-level web data extraction from template pages," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 2, pp. 249–263, Feb. 2010.
[22] D. E. Knuth, J. H. Morris Jr., and V. R. Pratt, "Fast pattern matching in strings," *SIAM J. Comput.*, vol. 6, no. 2, pp. 323–350, 1977.
[23] R. Kosala, H. Blockeel, M. Bruynooghe, and J. V. den Bussche, "Information extraction from structured documents using *k*-testable tree automaton inference," *Data Knowl. Eng.*, vol. 58, no. 2, pp. 129–158, Aug. 2006.
[24] N. Kushmerick, D. S. Weld, and R. B. Doorenbos, "Wrapper induction for information extraction," in *Proc. IJCAI*, 1997, pp. 729–737.
[25] B. Liu and Y. Zhai, "NET: A system for extracting web data from flat and nested data records," in *Proc. 6th Int. Conf. WISE*, New York, NY, USA, 2005, pp. 487–495.
[26] W. Liu, X. Meng, and W. Meng, "ViDE: A vision-based approach for deep web data extraction," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 3, pp. 447–460, Mar. 2010.
[27] A. Machanavajjhala, A. S. Iyer, P. Bohannon, and S. Merugu, "Collective extraction from heterogeneous web lists," in *Proc. 4th ACM Int. Conf. WSDM*, Hong Kong, China, 2011, pp. 445–454.
[28] I. Muslea. (1998). *RISE: Repository of Online Information Sources used in Information Extraction* [Online]. Available: http://www.isi.edu/info-agents/RISE
[29] I. Muslea, S. Minton, and C. A. Knoblock, "Hierarchical wrapper induction for semistructured information sources," *Auton. Agents Multi-Agent Syst.*, vol. 4, no. 1–2, pp. 93–114, Mar./Jun. 2001.
[30] L. Qian, M. J. Cafarella, and H. V. Jagadish, "Sample-driven schema mapping," in *Proc. 2012 ACM SIGMOD Conf.*, Scottsdale, AZ, USA, pp. 73–84.
[31] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures.* Boca Raton, FL, USA: Chapman and Hall/CRC, 2011.
[32] K. Simon and G. Lausen, "ViPER: Augmenting automatic information extraction with visual perceptions," in *Proc. 14th ACM Int. CIKM*, Bremen, Germany, 2005, pp. 381–388.
[33] H. A. Sleiman and R. Corchuelo, "A reference architecture to devise web information extractors," in *Proc. CAiSE Workshops*, Gdańsk, Poland, 2012, pp. 235–248.
[34] H. A. Sleiman and R. Corchuelo, "An unsupervised technique to extract information from semi-structured web pages," in *Proc. 13th Int. Conf. WISE*, Paphos, Cyprus, 2012, pp. 631–637.
[35] H. A. Sleiman and R. Corchuelo, "A survey on region extractors from web documents," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 9, pp. 1960–1981, Sept. 2012.
[36] H. A. Sleiman and R. Corchuelo, "TEX: An efficient and effective unsupervised web information extractor," *Knowl.-Based Syst.*, vol. 39, pp. 109–123, Feb. 2013.
[37] S. Soderland, "Learning information extraction rules for semi-structured and free text," *Mach. Learn.*, vol. 34, no. 1–3, pp. 233–272, Feb. 1999.
[38] W. Su, J. Wang, and F. H. Lochovsky, "ODE: Ontology-assisted data extraction," *ACM Trans. Database Syst.*, vol. 34, no. 2, Article 12, Jun. 2009.
[39] J. Wang and F. H. Lochovsky, "Data extraction and label assignment for web databases," in *Proc. 12th Int. Conf. WWW*, Budapest, Hungary, 2003, pp. 187–196.
[40] Y. Zhai and B. Liu, "Structured data extraction from the web based on partial tree alignment," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 12, pp. 1614–1628, Dec. 2006.

Hassan A. Sleiman received the Ph.D. degree from the University of Sevilla, Seville, Spain, in 2012, where he is currently a Lecturer with the Department of Computer Languages and Systems. Previously, he worked as a Software Engineer for companies such as Dynagent and set up a spin-off called Indevia. His current research interests include researching on web data extraction as a means to populate large datasets in the Web of data.

Rafael Corchuelo is a Reader of Software Engineering who is with the Department of Computer Languages and Systems of the University of Seville, Seville, Spain. He received the Ph.D. degree from the University of Seville, and has led the Research Group on Distributed Systems since 1997. His current research interests include integration of web data islands. Previously, he worked on advanced multi-party synchronisation models and fairness issues.