Knowledge-Based Systems 39 (2013) 109–123

Contents lists available at SciVerse ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

TEX: An efficient and effective unsupervised Web information extractor

Hassan A. Sleiman *, Rafael Corchuelo

Universidad de Sevilla, ETSI Informática. Avda. de la Reina Mercedes, s/n, Sevilla E-41012, Spain

ARTICLE INFO

Article history:
Received 23 March 2012
Received in revised form 11 October 2012
Accepted 12 October 2012
Available online 26 October 2012

Keywords:
Information extraction
Semi-structured web documents
Malformed documents
Unsupervised technique
Heuristic-based technique

ABSTRACT

The World Wide Web is an immense information resource. Web information extraction is the task that transforms human friendly Web information into structured information that can be consumed by automated business processes. In this article, we propose an unsupervised information extractor that works on two or more web documents generated by the same server side template. It finds and removes shared token sequences amongst these web documents until finding the relevant information that should be extracted from them. The technique is completely unsupervised and does not require maintenance, it allows working on malformed web documents, and does not require the relevant information to be formatted using repetitive patterns. Our complexity analysis reveals that our proposal is computationally tractable and our empirical study on real-world web documents demonstrates that it performs very fast and has a very high precision and recall.

1. Introduction

The Web is the hugest information repository. Usually, scripts are used to fill in templates with information that is retrieved from server-side databases; the results are formatted using HTML tags and CSS classes. The documents in the Web can be classified into two groups: unstructured documents, whose relevant information are pieces of free text, e.g., blog entries or news articles, and semi-structured documents, whose relevant information are records of attributes that are usually formatted as tables or lists, e.g., an online book store. (Note that the criteria to classify which information is relevant depends completely on the context.) Our work focuses on semi-structured documents.

Extracting the relevant information from a semi-structured document to feed an automated business process is not usually an easy task due to the irrelevant information that the template introduces in order to present it in a friendly format [3]. Information extractors are intended to help software engineers in this task [10].

Many information extractors rely on extraction rules. Although they can be handcrafted [15,24,4,42,51,50,20], the costs involved motivated many researchers to work on proposals to learn them automatically. These proposals are either supervised, i.e., they require the user to provide a number of information samples to be extracted [11,44,58,26,32,8,22,9,14,18,30,5,40,21,59], or unsupervised, i.e., they extract as much prospective information as they can and the user then gathers the relevant information from the results [62,12,16,2,28,25,60,39,46,64,67,38,59,57]. Since typical web documents are growing in complexity, a number of authors are also working on techniques whose goal is to identify the region within a web document where relevant information is most likely to be contained [37,7,61,63,27,34,65,66,52,45,35,6]. Sleiman and Corchuelo [55] have recently surveyed and compared the previous techniques.

Information extractors that rely on extraction rules do not usually adapt well to changes to the Web. Note that once a set of extraction rules is handcrafted or learnt, the Web keeps evolving and it is not uncommon that changes may invalidate the existing extraction rules. This motivated some authors to work on proposals to maintain extraction rules (semi-)automatically [31,48,49,41,33,13]. Contrarily, others worked on unsupervised proposals that do not rely on extraction rules, but are based on a number of hypothesis and heuristics that have proven to work well in many cases [1,53,17,23]; changes to a web site do not usually have an impact on these extractors since they analyse every new web document independently from the previous ones.

Our focus is on unsupervised proposals that do not rely on extraction rules. The existing proposals work on one or more input web document and search for repetitive structures that hopefully identify the regions where the relevant information resides. Álvarez et al. [1] use clustering to find a rough region where the relevant information is most likely to be located, i.e., the information region, and then use clustering, tree matching and multi-string alignment to extract prospective information; Simon and Lausen [53] first use a modified version of MDR [36] and then a multi-string alignment algorithm to extract prospective information; Buttler et al. [6] rely on six heuristics to identify the information region and to extract prospective information from it; the proposals by Refs. [17,23] focus on extracting prospective information from lists:

* Corresponding author.
E-mail addresses: hassansleiman@us.es (H.A. Sleiman), corchu@us.es (R. Corchuelo).

the former uses a corpus and a scoring function that helps delimit the information in a list and tabulate it, whereas the latter learns a statistical model according to which the information is also delimited and tabulated. Implicitly, the previous proposals assume that the input web documents contain similar information records since they all rely on finding repetitive structures.

A four-page abstract of our proposal was presented elsewhere [56]. In this article, we introduce TEX, which is an unsupervised information extractor that does not rely on extraction rules. Contrarily to the previous proposals, it does not require the input web documents to be translated into DOM trees, i.e., it can work on malformed web documents without correcting them, and does not require the relevant information to be formatted using repetitive structures inside a web document. It works on two or more web documents and compares them in an attempt to discover shared patterns that are not likely to provide any relevant information, but parts of the template used to generate the web documents. (We define a shared pattern of size *s* between two token sequences *t*1 and *t*2 as a subsequence of *s* consecutive tokens that occurs at least once in both *t*1 and *t*2.) The idea of identifying shared patterns lies at the heart of proposals like RoadRunner [16], which uses a multi-string alignment algorithm to learn a regular expression that models the template and its variable parts, FiVaTech [28], which relies on tree matching, tree alignment, and mining techniques, or EXALG [2], which uses two statistical techniques to differentiate the role of individual tokens and determine which are equivalent to one another. Contrarily to these proposals, TEX relies on quite a simple multi-string alignment algorithm that has proven to be very effective and efficient in practice. We have computed an upper limit to the worst-case space and time complexity of our algorithm and we have proven that it is computationally tractable (note that there are very few complexity results in this field); furthermore, we have conducted a series of experiments with 2084 web documents from 55 real-world web sites and our results confirm that our proposal can achieve a mean precision as high as 96%, a mean recall as high as 95%, with a mean execution time of 0.81 s. We conducted the same experiments using other well-known techniques in the literature, and our conclusion is that our proposal outperforms them.

The rest of the article is organised as follows: Section 2 presents TEX and describes the sub-algorithms on which it relies; Section 3 analyses its time complexity; Section 4 reports on our experimental results and compares TEX to other techniques in the literature; finally, Section 5 concludes our work.

2. Description of our proposal

We present the algorithm that lies at the heart of TEX in Fig. 1. It works on a collection of web documents, which we denote as TextSet, and a range of integers, which can introduce a bias to our search procedure. Intuitively, a TextSet is a set of Texts, which are sequences of Tokens. TEX is not bound with a particular tokenisation schema; our implementation and our experiments were carried out using a simple tokenisation schema according to which tokens represent either script blocks, style blocks, HTML tags, or #PCDATA, but this is not an intrinsic feature of our proposal. Note that we use Text as a data type that allows to represent both web documents and fragments of web documents, as well as the information that is extracted from them.

```
1: TEX ($ts$: TextSet; $min$, $max$: int): List⟨TextSet⟩
2:     $l$ = extract($ts$, $min$, $max$)
3:     $result$ = filter($l$)
4: return $result$
```

Fig. 1. Algorithm TEX.

The algorithm works in two steps: at line 2, we invoke Algorithm extract, which makes an attempt to extract the information that varies from document to document; in other words, it attempts to discard information that is likely to belong to the template used to generate the input web documents. Algorithm extract works on the collection of input web documents and searches for shared patterns of size $max, max-1, \ldots, min$. If $min > 1$ or *max* is less than the size of the shortest input document, then the search has a bias that may lead to situations in which Algorithm extract returns information that actually belongs to the template, which is the reason why we invoke a filtering algorithm at line 3.

Fig. 2 presents a running example. We assume that the algorithm is executed on TextSet *TS*1, which is composed of documents *T*1, *T*2, and *T*3; the result is the list of TextSets *L*1, which contains the extracted TextSets *TS*4, *TS*7, *TS*11, *TS*12, *TS*9, and *TS*10.

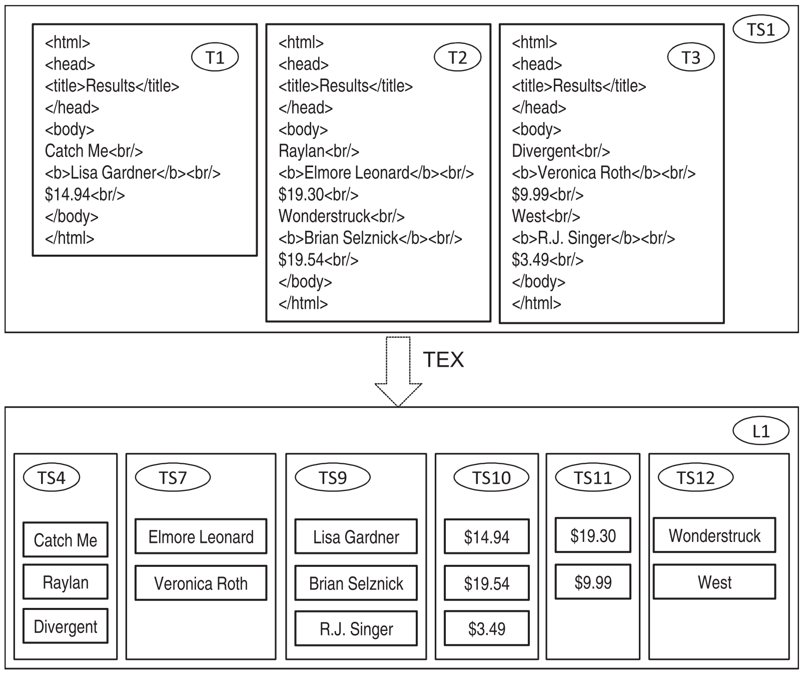
In the following subsections, we provide additional details on the ancillary algorithms on which TEX relies.

2.1. Algorithm extract

Algorithm extract searches for shared patterns of size *max* down to *min* in a TextSet. For instance, assume that it is invoked on the TextSet denoted as *TS*1 in Fig. 3 and that it has to search for shared patterns whose size is in the range 10 down to 1. Note that there are neither shared patterns of size 10, 9, nor 8; the longest shared pattern is <html><head><title>Results</title></Head><body>, whose size is 7 tokens. The algorithm then attempts to expand TextSet *TS*1 into three additional TextSets that contain the prefixes, the separators, and the suffixes into which the shared pattern partitions the Texts in *TS*1. In this example, there are neither prefixes nor separators, since the shared pattern is found at the beginning of the Texts in *TS*1; there are, however, three suffixes that are stored in TextSet *TS*2. The algorithm then discards TextSet *TS*1 and proceeds with the new TextSet *TS*2. The longest shared pattern that is discovered in *TS*2 is <br/></body></html>, which results in a new TextSet that is denoted as *TS*3. The same procedure is applied as many times as necessary until no more shared patterns are discovered.

We present Algorithm extract in Fig. 4. It works on a TextSet *ts*, a minimum pattern size *min* and a maximum pattern size *max*; it returns a list of TextSets that should contain as much prospective information as possible. The main loop at lines 3–15 iterates over all possible sizes from *max* down to *min*; for each size, the inner loop at lines 5–13 searches for a shared pattern of that size. Note that variable *result* acts as a queue in which we initially put the TextSet on which the algorithm has to work, and then the new TextSets into which it is expanded. In each iteration of the inner loop, a TextSet is removed from *result* and expanded at line 7. Algorithm expand, which is presented in the following section, searches for shared patterns of a given size in a TextSet; if one such pattern is found, then it is used to expand the current TextSet into new TextSets with prefixes, separators, and suffixes, which are added to *result* so that they can be analysed later in the inner loop; if no shared pattern is found, then the original TextSet is added to a buffer. Once the inner loop finishes, the buffer contains all of the new TextSets that have been produced, and it is transferred to the *result* variable so that the algorithm can search for new shared patterns of a smaller size, if possible.

Algorithm expand. This algorithm searches for a shared pattern of a given size inside a given TextSet; if such a pattern is found, it then expands the TextSet into a collection of new TextSets with prefixes, separators, and suffixes. We have already illustrated how Algorithm expand works on two simple cases in which the expansion

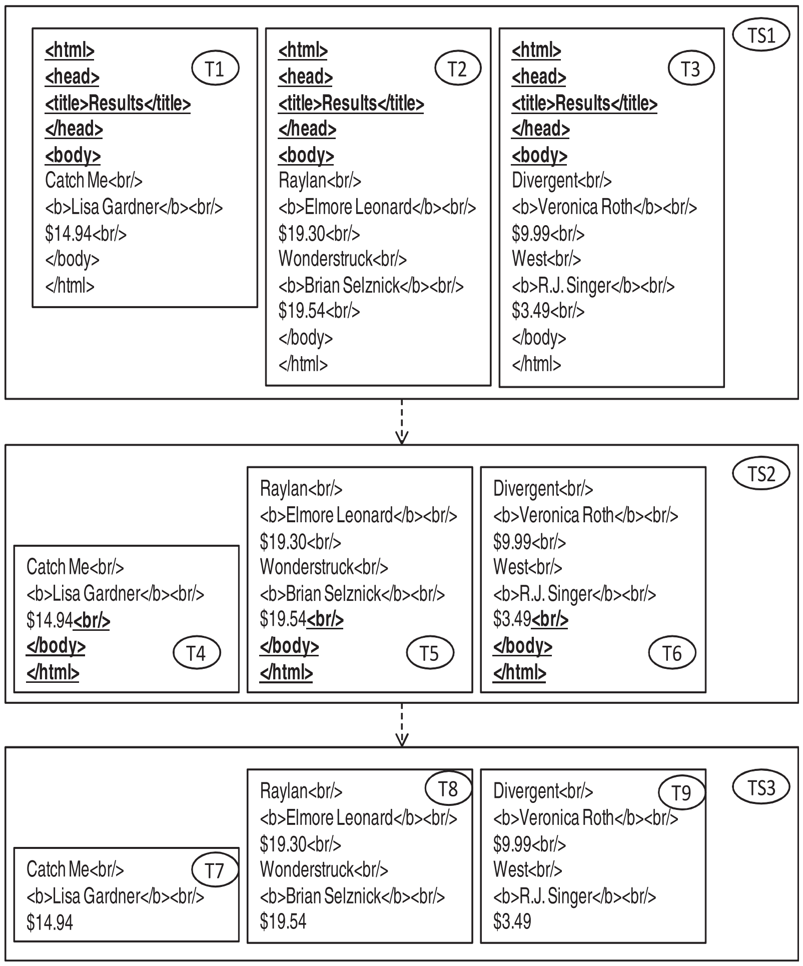


```
<html>
<head>
<title>Results</title>
</head>
<body>
Catch Me<br/>
<b>Lisa Gardner</b><br/>
$14.94<br/>
</body>
</html>
```

```
<html>
<head>
<title>Results</title>
</head>
<body>
Raylan<br/>
<b>Elmore Leonard</b><br/>
$19.30<br/>
Wonderstruck<br/>
<b>Brian Selznick</b><br/>
$19.54<br/>
</body>
</html>
```

```
<html>
<head>
<title>Results</title>
</head>
<body>
Divergent<br/>
<b>Veronica Roth</b><br/>
$9.99<br/>
West<br/>
<b>R.J. Singer</b><br/>
$3.49<br/>
</body>
</html>
```

Fig. 2. A running example.



```
<html>
<head>
<title>Results</title>
</head>
<body>
Catch Me<br/>
<b>Lisa Gardner</b><br/>
$14.94<br/>
</body>
</html>
```

```
<html>
<head>
<title>Results</title>
</head>
<body>
Raylan<br/>
<b>Elmore Leonard</b><br/>
$19.30<br/>
Wonderstruck<br/>
<b>Brian Selznick</b><br/>
$19.54<br/>
</body>
</html>
```

```
<html>
<head>
<title>Results</title>
</head>
<body>
Divergent<br/>
<b>Veronica Roth</b><br/>
$9.99<br/>
West<br/>
<b>R.J. Singer</b><br/>
$3.49<br/>
</body>
</html>
```

```
Catch Me<br/>
<b>Lisa Gardner</b><br/>
$14.94<br/>
</body>
</html>
```

```
Raylan<br/>
<b>Elmore Leonard</b><br/>
$19.30<br/>
Wonderstruck<br/>
<b>Brian Selznick</b><br/>
$19.54<br/>
</body>
</html>
```

```
Divergent<br/>
<b>Veronica Roth</b><br/>
$9.99<br/>
West<br/>
<b>R.J. Singer</b><br/>
$3.49<br/>
</body>
</html>
```

```
Catch Me<br/>
<b>Lisa Gardner</b><br/>
$14.94
```

```
Raylan<br/>
<b>Elmore Leonard</b><br/>
$19.30<br/>
Wonderstruck<br/>
<b>Brian Selznick</b><br/>
$19.54
```

```
Divergent<br/>
<b>Veronica Roth</b><br/>
$9.99<br/>
West<br/>
<b>R.J. Singer</b><br/>
$3.49
```

Fig. 3. Expansion of a TextSet during extraction.

```
extract($ts$: TextSet; $min$, $max$: int): List⟨TextSet⟩
    $result = \langle ts \rangle$
    for $size = max$ down to $min$ do
        $buffer = \langle\rangle$
        while $result \neq \langle\rangle$ do
            $ts$ = dequeue($result$)
            $expansion$ = expand($ts$, $size$)
            if $expansion = \langle\rangle$ then
                enqueue($buffer$, $ts$)
            else
                enqueue($result$, $expansion$)
            end
        end
        $result = buffer$
    end
return $result$
```

Fig. 4. Algorithm extract.

led to prefixes or suffixes only, cf. Fig. 3. Assume now that it is invoked on TextSet *TS*3 in Fig. 5 to search for a shared pattern of size two tokens. The algorithm can easily detect that the first two-token shared pattern is <br/><b> and expands TextSet *TS*3 into the following new TextSets: (i) *TS*4, which contains the prefixes of the Texts in *TS*3 up to the first occurrence of the shared pattern; (ii) *TS*5, which contains the separators, i.e., the Texts in *TS*3 in between successive occurrences of the shared pattern; (iii) and *TS*6, which contains the suffixes of the Texts in *TS*3 regarding the last occurrence of the shared pattern. When expand is invoked on TextSet *TS*5 to find a shared pattern of size two tokens, it finds </b><br/>, and creates *TS*7 and *TS*8, which contain the prefixes and suffixes of the shared pattern respectively. The same happens when expand is invoked on the TextSet *TS*6 to search for a shared pattern of size two tokens, and creates *TS*9 and *TS*10. If we invoke expand on TextSet *TS*8 to search for a shared pattern of size one token, then, it finds pattern <br/> and creates TextSets *TS*11 and *TS*12.

We present Algorithm expand in Fig. 6. Line 3 searches for the shortest Text in *ts*, which is used at line 5 as a basis to search for shared patterns by means of Algorithm findPattern, which is described in the following section. If this algorithm can find a shared pattern, then line 7 expands *ts* using that shared pattern and updates variable *result*; if not, *result* remains an empty list, which indicates that TextSet *ts* cannot be expanded.

Algorithm findPattern. This algorithm works on a TextSet *ts*, a Text *base*, which is assumed to be the shortest non-empty Text in *ts*, and a size *s*. Its goal is to find a pattern inside *base* that occurs in every Text in *ts*. For instance, assume that the algorithm is invoked on TextSet *TS*3 in Fig. 7 to search for a pattern of size 2; we implicitly assume that *base* is the shortest Text in *TS*3, i.e., *base* = Catch Me <b> Lisa Gardner </b> $14.94 in this example. The algorithm first searches for Catch Me<br/> in every Text in *TS*3, but does not find it; then it searches for <br/><b>, which is found in every Text in *TS*3. As a conclusion <br/><b> is a shared pattern that can be used to expand TextSet *TS*3. Note that Algorithm findPattern returns a map from Text onto lists of integers; the map represents the positions where the search pattern is found. In our example, this map is $\{T7 \mapsto \langle 1 \rangle, T8 \mapsto \langle 1, 9 \rangle, T9 \mapsto \langle 1, 9 \rangle\}$.

We present Algorithm findPattern in Fig. 8. The main loop at lines 3–11 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, i.e., it searches for all patterns of size *s* in *base*. The actual search is performed in the inner loop at line 6–10: in this loop, the algorithm iterates over every Text in the input TextSet and finds all of the matches of the subsequence of *base* that starts at position *i* and has size *s*. We do not provide any additional details on Algorithm findMatches since it is implemented using the well-known Knuth–Pratt–Morris pattern search algorithm [29]. This algorithm

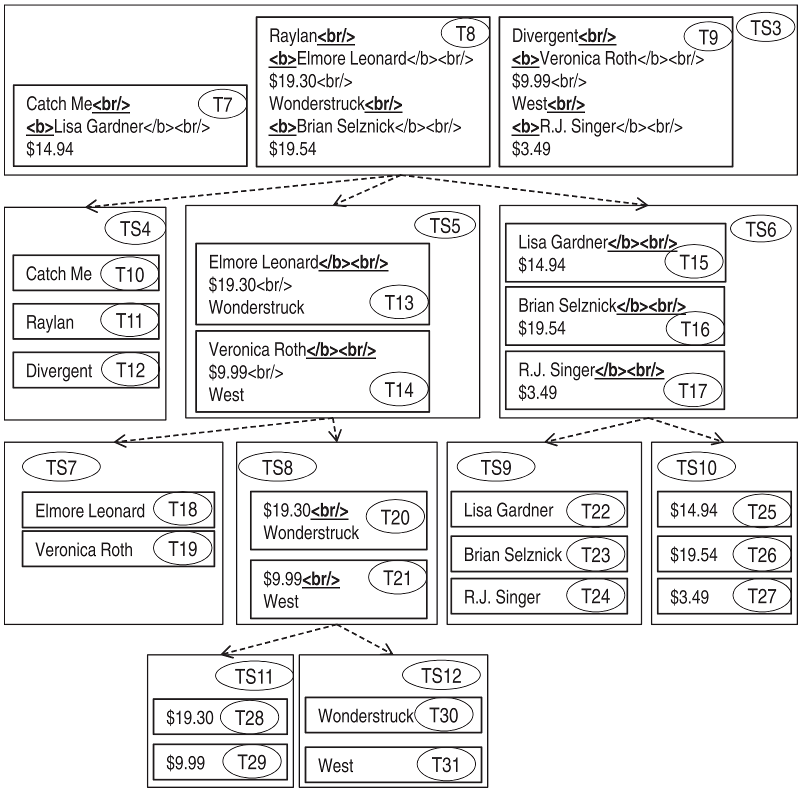


Fig. 5. Expansion of a sample TextSet.

```
1: expand($ts$: TextSet; $s$: int): List⟨TextSet⟩
2:     $result = \langle\rangle$
3:     $shortest$ = find the shortest text in $ts$
4:     if $shortest \neq \langle\rangle$ and size($shortest$) >= $s$ then
5:         $shared$ = findPattern($ts$, $shortest$, $s$)
6:         if $shared \neq \{\}$ then
7:             $result$ = createExpansion($ts$, $shared$)
8:         end
9:     end
10: return $result$
```

Fig. 6. Algorithm expand.

returns a list of integers 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 *ts*; otherwise, the inner loop finishes and the outer loop slides the window on *base* and resets *result*, if possible. If the algorithm returns an empty map, this means that no shared pattern has been found.

Algorithm createExpansion. When a shared pattern is found in a TextSet, the TextSet is expanded to three new TextSets, namely: prefixes, separators, and suffixes. We present Algorithm createExpansion in Fig. 9. This algorithm works on a TextSet called *ts* and a map *r* that contains the indexes of the shared pattern inside each Text of *ts*. The loop at lines 6–15 iterates over all of the Texts in *ts* and adds the prefixes, separators, and suffixes to variables ts_1, ts_2, and ts_3, respectively. Later, we add these intermediate TextSets to the *result* variable as long as they are not empty.

2.2. Algorithm filter

Algorithm extract returns a list of TextSets that are expected to have the variable information in the initial TextSet. Note, however, that *min* and *max* introduce a bias to the search algorithm if *min* is greater than one or *max* is less than the size of the shortest Text in the initial TextSet. Our experimental results prove that this bias helps effectively reduce the amount of effort required to extract information from typical web documents, without sacrificing effectiveness. There are, however, cases in which setting *min* to a value greater than one and setting *max* to a small value, may prevent Algorithm extract from finding small shared patterns. For instance, assume that we set $min = 2$ and $max = 2$ and that Algorithm extract returns the list of TextSets $L'1$ in Fig. 10: if the algorithm was allowed to search for patterns of size one, then it would discover that <html> is a shared pattern and would discard TextSet *TS*′8; however, *min* was set to 2, which prevents the algorithm from finding this pattern.

We present Algorithm filter in Fig. 11. The main loop at lines 3–7 iterates over the list of input TextSets and simply removes those without variability from the result, i.e., those TextSets in which all of the Texts are the same.

3. Complexity analysis

In this section, we provide an upper limit to the worst-case time complexity of Algorithm TEX. Note that it is not common to find a complexity analysis in the literature regarding information extraction, but we think that it is important to make sure that the proposal is computationally tractable. Note, too, that the actual time complexity depends on a variety of variables that are not easy

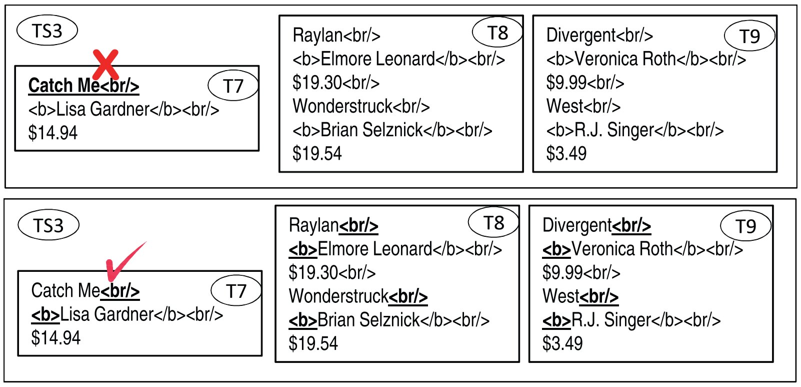

TS3
Catch Me
 <b>Lisa Gardner</b>
 $14.94
T7
Raylan<br/> <b>Elmore Leonard</b><br/> $19.30<br/> Wonderstruck<br/> <b>Brian Selznick</b><br/> $19.54
T8
Divergent<br/> <b>Veronica Roth</b><br/> $9.99<br/> West<br/> <b>R.J. Singer</b><br/> $3.49
T9
TS3
Catch Me
 <b>Lisa Gardner</b>
 $14.94
T7
Raylan<br/> <b>Elmore Leonard</b><br/> $19.30<br/> Wonderstruck<br/> <b>Brian Selznick</b><br/> $19.54
T8
Divergent<br/> <b>Veronica Roth</b><br/> $9.99<br/> West<br/> <b>R.J. Singer</b><br/> $3.49
T9

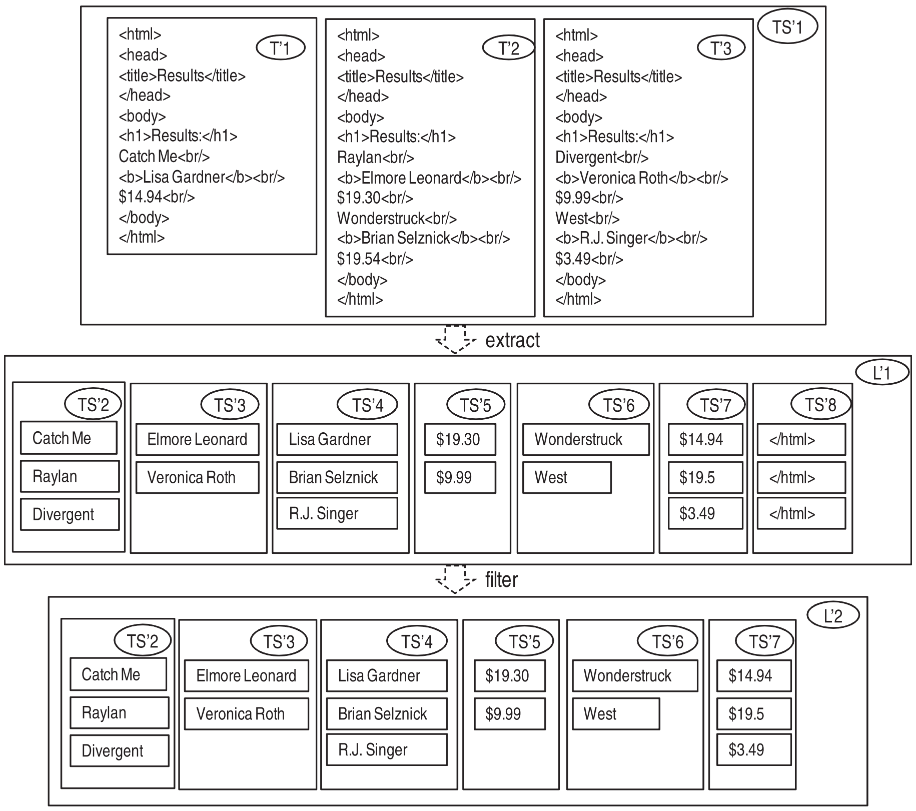
Fig. 7. Searching for a pattern using Catch Me <b> Lisa Gardner </b> $14.94 as a base.

```
1: findPattern($ts$: TextSet; $base$: Text; $s$: int): Map<Text, List⟨int⟩>
2:     $found$ = false
3:     for $i = 0$ until size($base$) - $s$ while not $found$ do
4:         $result = \{\}$
5:         $found$ = true
6:         foreach $text$ in $ts$ while $found$ do
7:             $matches$ = findMatches($text$, $base$, $i$, $s$)
8:             $found$ = size($matches$) > 0
9:             $result = result \cup \{text \mapsto matches\}$
10:        end
11:    end
12: return $result$
```

Fig. 8. Algorithm findPattern.

```
createExpansion($ts$: TextSet; $r$: Map⟨Text, List⟨int⟩⟩; $s$: int): List⟨TextSet⟩
    $result = \langle\rangle$
    $ts_1$ = new TextSet()
    $ts_2$ = new TextSet()
    $ts_3$ = new TextSet()
    foreach $text$ in $ts$ do
        $matches$ = get($r$, $text$)
        if prefix($matches$, $text$) $\neq \langle\rangle$ then
            add prefix($matches$, $text$) to $ts_1$
        end
        add non-empty separators($matches$, $text$) to $ts_2$
        if suffix($matches$, $text$) $\neq \langle\rangle$ then
            add suffix($matches$, $text$) to $ts_3$
        end
    end
    if $ts_1$ is not empty then
        add $ts_1$ to $result$
    end
    if $ts_2$ is not empty then
        add $ts_2$ to $result$
    end
    if $ts_3$ is not empty then
        add $ts_3$ to $result$
    end
return result
```

Fig. 9. Algorithm createExpansion.



```
<html>
<head>
<title>Results</title>
</head>
<body>
<h1>Results:</h1>
Catch Me<br/>
<b>Lisa Gardner</b><br/>
$14.94<br/>
</body>
</html>
```

```
<html>
<head>
<title>Results</title>
</head>
<body>
<h1>Results:</h1>
Raylan<br/>
<b>Elmore Leonard</b><br/>
$19.30<br/>
Wonderstruck<br/>
<b>Brian Selznick</b><br/>
$19.54<br/>
</body>
</html>
```

```
<html>
<head>
<title>Results</title>
</head>
<body>
<h1>Results:</h1>
Divergent<br/>
<b>Veronica Roth</b><br/>
$9.99<br/>
West<br/>
<b>R.J. Singer</b><br/>
$3.49<br/>
</body>
</html>
```

Fig. 10. A case in which filtering the results of Algorithm extract is necessary.

to characterise; since our only goal was to prove that TEX is computationally tractable, we have just characterised an upper bound to the worst-case time complexity building on two sensible assumptions: (i) simple instructions like adding an item to a set, comparing

```
1: filter($tss$: List⟨TextSet⟩): List⟨TextSet⟩
2:     $result$ = ⟨⟩
3:     foreach $ts$ in $tss$ do
4:         if $ts$ has variability then
5:             add $ts$ to $result$
6:         end
7:     end
8: return result
```

Fig. 11. Algorithm filter.

two tokens, or constructing a tuple can be implemented in $O(1)$ time with regard to the other algorithms in our proposal; (ii) the number of input documents is generally very small as compared with the number of tokens of the longest document to be analysed.

In the following subsections, we first present a preliminary result regarding space requirements, then analyse the time complexity of the ancillary algorithms on which TEX relies, and conclude with a theorem that proves that TEX is computationally tractable. In the sequel, we use variable n to denote the number of documents that TEX has to analyse and m to denote the size in tokens of the longest such document.

3.1. Space requirements

Proposition 1 (*Maximum size of a TextSet*). *Assume that we are extracting information from a* TextSet *denoted as ts using Algorithm TEX.* $n\lfloor\frac{m}{2}\rfloor$ *is an upper bound to maximum size of a* TextSet *generated by TEX.*

Proof. Algorithm expand is the unique algorithm that generates new TextSets, which happens when a shared pattern p is found in a given TextSet. The new TextSets correspond to the prefixes, separators, and suffixes to which p leads. Regarding the prefix and suffix TextSets, note that there cannot be more than n such prefixes or suffixes; that is, n is an upper bound to the maximum size of a TextSet that contains prefixes or suffixes. Regarding separator TextSets 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, or, otherwise, $n\lfloor\frac{m}{2}\rfloor$ is an upper bound to the maximum size of a TextSet that contains separators. As a conclusion, $n\lfloor\frac{m}{2}\rfloor$ is an upper bound to the maximum size of a TextSet generated by TEX. □

3.2. Time requirements

Lemma 1 (*Algorithm createExpansion*). *Let ts be a* Textset, *r be a map from the* Texts *in ts to lists of indices that denote where a shared pattern occurs, and s be the size of the shared pattern.* $O(nm^2)$ *is an upper bound to the worst-case time required to execute* createExpansion(ts, r, s).

Proof. The algorithm iterates through every Text in *ts*. According to Proposition 1, $n\lfloor\frac{m}{2}\rfloor$ is an upper bound to the maximum size of a TextSet, which means that the nm 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 Proposition 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. As a conclusion, $O(nmm) = O(nm^2)$ is an upper bound to the worst-case time required to execute createExpansion(ts, r, s). □

Lemma 2 (*Algorithm findPattern*). *Let ts be a* TextSet, *base be the shortest* Text *in ts, and s be the size of the pattern for which the algorithm searches. Then,* $O(nm^3)$ *is an upper bound to the worst-case time required to execute* findPattern$(ts, base, s)$.

Proof. The main loop iterates through *base* until finding a pattern of *s* tokens that occurs in every other Text in *ts*. 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 *ts*. According to Proposition 1, $n\lfloor\frac{m}{2}\rfloor$ is an upper bound to the maximum size of a TextSet, 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 [29]. 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, $O(mn\lfloor\frac{m}{2}\rfloor m) \subseteq O(nm^3)$ is an upper bound to the worst-case time required to execute findPattern$(ts, base, s)$. □

Lemma 3 (*Algorithm expand*). *Let ts be a* TextSet, *and s be a pattern size.* $O(nm^3)$ *is an upper bound to the worst-case time required to execute* expand(ts, s).

Proof. The algorithm first searches for the shortest Text in *ts*. According to Proposition 1, $n\lfloor\frac{m}{2}\rfloor$ is an upper bound to the maximum size of a TextSet, which implies that $n\lfloor\frac{m}{2}\rfloor$ is also an upper bound to the maximum time required to find the shortest Text in a TextSet. In the worst case, the invocation to Algorithm expand requires to invoke Algorithms findPattern and createExpansion in sequence, which according to Lemmas 2 and 1 require no more than $O(nm^3)$ and $O(nm^2)$ time in the worst case. As a conclusion, $O(n\lfloor\frac{m}{2}\rfloor + nm^3 + nm^2) \subseteq O(nm^3)$ is an upper bound to the worst-case time required to execute expand(ts, s). □

Lemma 4 (*Algorithm extract*). *Let ts be a* TextSet, *min and max be the minimum and maximum sizes of the shared patterns for which the algorithm searches, respectively.* $O(nm^5)$ *is an upper bound to the worst-case time required to execute* extract(ts, min, max).

Proof. The algorithm first iterates through all possible sizes between *min* and *max*, which amounts to *m* times in the worst case. In each iteration, the algorithm executes an inner loop that iterates through successive expansions of *ts*. Note that *m* puts an upper bound to the number of times that a TextSet can be expanded, which implies that *m* is also an upper bound to the number of iterations of the inner loop. Within this loop, the only significant instruction regarding our complexity analysis is the invocation of Algorithm expand, which according to Lemma 3 requires no more than $O(nm^3)$ time. As a conclusion, $O(mmnm^3) = O(nm^5)$ is an upper bound to the worst-case time required to execute extract(ts, min, max). □

Lemma 5 (*Algorithm filter*). *Let L be a list of* TextSets *of size k.* $O(knm)$ *is an upper bound to the worst-case time required to execute* filter(L).

Proof. The algorithm iterates through every Text in *L*. If we denote the size of *L* as *k*, then this loop iterates *k* times. In each iteration, the algorithm checks the variability of the current TextSet, which requires to compare the first Text to every other in order to determine whether the TextSet has variability or not. According to

Proposition 1, $n\lfloor\frac{m}{2}\rfloor$ is an upper bound to the number of Texts in a TextSet, which implies that $O(kn\lfloor\frac{m}{2}\rfloor) \subseteq O(knm)$ is an upper bound to the worst-case time required to execute filter(L). □

3.3. Complexity of TEX

Theorem 1 (*Algorithm TEX*). *Let ts be a* TextSet *and min and max be the minimum and maximum sizes of the shared patterns for which TEX searches. Then, $O(nm^5)$ is an upper bound to the worst-case time required to execute TEX(ts, min, max).*

Proof. The proof follows straightforwardly from the previous lemmas. Note that Algorithm TEX invokes Algorithms extract and filter in sequence, which, according to Lemmas 4 and 5 require no more than $O(nm^5)$ and $O(knm)$ time to complete, where k denotes the size of the list of TextSets returned by Algorithm extract. Note that m puts an upper bound to the size of this list, which implies that $O(nm^5 + nm^2) \subseteq O(nm^5)$ is an upper bound to the worst-case time required to execute TEX(ts, min, max). □

Corollary 1. *If we assume that $n \ll m$, then $O(m^5)$ is an upper bound to the worst-case time complexity of Algorithm* TEX, *which makes it computationally tractable.*

4. Experimental results

In this section, we present the results of the experiments we have carried out to compare TEX to other techniques in the literature from an empirical point of view. We have developed a prototype of TEX using the CEDAR framework [54]. We performed our experiments on a machine that was equipped with an Intel Core i7 processor that ran at 3.4 GHz, had 8 GB of RAM, Windows 7 Pro 64-bit, Oracle's Java Development Kit 1.7.0_03_b02, and GNU Regex 1.1.4.

First, we describe the datasets used in our experimental study, then we describe the techniques in the literature with which we compared TEX, next we study the effectiveness and the efficiency of the techniques, and finally we rank them statistically.

4.1. Datasets

We performed our experiments on a collection of 55 datasets that contain a total of 2084 web documents that can be classified into two groups: the first group is composed of 41 datasets gathered from 41 real-world web sites, whereas the second group is composed of 14 datasets downloaded from two 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, i.e., we created a number of TextSets with the information that we would like to extract from each site. The second group contains all of the datasets available online at the EXALG repository [2] and the datasets composed of semi-structured web documents available at the RISE repository [43]. Table 1 show the information in which we were interested from the datasets.

JTidy is a Java implementation of HTML tidy [47], which is a proposal that is intended to preprocess web documents by fixing their HTML code and converting it into XHTML. For instance, it fixes web document doctype declarations, adds missing end tags, and reports on unknown attributes, if necessary. Since our datasets were gathered from real-world web sites, they usually contained errors in their HTML code. Table 2 presents the results we have gathered regarding a subset of common HTML errors that are reported by JTidy; the full report is too large to be reproduced here. Our only purpose was to make it clear that we have dealt with actual documents, and that they usually contain errors that must be fixed heuristically. JTidy is a constituent part of many information extraction proposals that build on DOM trees. Note that TEX does not require to use JTidy since it can work on malformed input documents.

Table 3 reports on some properties of the datasets. The first column illustrates the categories of the datasets; the second and the third columns list the web sites and an identifier that we use in the following tables; the fourth column reports on the number of documents inside each dataset; the fifth column shows the mean size of documents in kibibytes; the sixth column shows the mean number of errors reported by JTidy inside each dataset; the seventh column reports on the mean time in seconds required to clean and fix a web document; finally, the last column reports on the mean time to tokenise each document.

4.2. Other techniques

We searched the Web or requested the authors of the related information extraction techniques to send us the implementations of their proposals. Unfortunately, we only managed to get the following implementations:

- RoadRunner [16]: It takes two or more web documents as input and tries to learn a union-free regular expression that describes them. It considers the first web document as a base template and then iterates through the other web documents; in each iteration, it compares the current web document with the base template using a string alignment algorithm, then collapses mismatches, and applies a backtracking algorithm to detect optional and repetitive patterns.
- FiVaTech [28]: It takes one or more web documents as input and tries to learn the template that was used to generate them. FiVaTech applies a clustering algorithm that applies a tree-edit distance to the DOM nodes of the input web documents; it then uses a matrix alignment algorithm to align the previous nodes on a per-cluster basis; then, it applies an algorithm to mine repetitive patterns in the aligned matrix, and, finally, applies some heuristics to detect optionality.
- SoftMealy [26]: It takes a collection of web documents and their corresponding annotations as input and learns an extraction rule that is a non-deterministic finite-state transducer. The states of the transducer indicate the attributes to extract, the transitions account for the possible orderings of the attributes in the input web documents, and the conditions indicate when the extraction of an attribute should start or end. Transition conditions are learnt using a token alignment and generalisation algorithm.
- WIEN [32]: It takes a collection of annotated web documents as input and learns simple regular expressions that contain the delimiters of the information that should be extracted. These delimiters are the longest common prefix of characters, and the longest common suffix of characters for each type of attributes. WIEN cannot handle optional attributes.

4.3. Effectiveness analysis

We first ran RoadRunner, FiVaTech, SoftMealy, and WIEN on the datasets in order to learn extraction rules. Since there is not an

Table 1
Information of interest from the datasets.

Repository	Category	Information of interest
Ours	Books	Title, author, price, year, isbn
	Cars	Model, year, description, price, type, colour, milage, transmission, engine, doors
	Conferences	Title, date, place, url
	Doctors	Name, address, phone, fax, specialty
	Jobs	Location, company
	Movies	Title, director, actor, year, runtime
	Real estate	Address, bedrooms, bathrooms, price, size
	Sports	Name, birth, hight, weight, age, college, country, club, position
EXALG	cars.amazon.com	Model, make, price
	players.uefa.com	Name, country
	popartist.amazon.com	Name
	teams.uefa.com	Fifa-affiliation, founded, general-secretary, president, press-officer, uefa-affiliation
	ausopen.com	Birthdate, birthplace, country, height, money, name, weight
	ebay.com	Price, bids, location
	majorleaguebaseball.com	Name, position, team
	netflix.com	Title, director, length, year
	rpmfind.net	Name, description, operative-system
RISE	bigbook.com	Name, city, phone, street
	iaf.net	Name, email, organisation, service-provider
	laweekly.com/restaurants	Name, address, speciality
	okra.ucr.edu	Name, email
	zagat.com	Name, address, type

Table 2
Subset of common errors.

Error	Mean
Error: <> is not recognised!	0.16 ± 0.57
Error: <> missing '>' for end of tag	0.41 ± 1.07
Error: discarding unexpected <>	1.50 ± 6.46
Error: missing quotemark for attribute value	0.01 ± 0.10
Warning: <> element not empty or not closed	24.31 ± 42.45
Warning: <> is not approved by W3C	2.70 ± 6.14
Warning: <> is not allowed after elements	0.03 ± 0.17
Warning: <> is not allowed in <> elements	13.71 ± 11.14
Warning: <> should not be nested	0.07 ± 0.46
Warning: <> unexpected or duplicate quote mark	3.65 ± 30.00
Warning: adjacent hyphens within comment	1.32 ± 2.29
Warning: discarding unexpected <>	3.33 ± 6.12
Warning: inserting implicit <>	1.23 ± 4.95
Warning: link is not allowed in <> elements	0.00 ± 0.02
Warning: meta is not allowed in <> elements	0.06 ± 0.45
Warning: missing <> before <>	3.21 ± 7.36
Warning: missing <> declaration	1.00 ± 0.00
Warning: missing <>	4.68 ± 8.02
Warning: plain text is not allowed in <> elements	0.74 ± 0.45
Warning: replacing element <> by <>	0.09 ± 0.29
Warning: replacing unexpected <> by <>	0.21 ± 1.01

upper bound to the learning time, we set a time out of 15 CPU minutes. We then computed precision (P), recall (R), and F_1-measure (F_1).

In the case of SoftMealy and WIEN, it was easy to compute the precision and recall for each attribute since both techniques are supervised, i.e., they require the user to provide annotations with the information to be extracted so that an extraction rule can be learnt and validated. Contrarily, TEX, RoadRunner, and FiVaTech are unsupervised, i.e., they extract as much prospective information as possible, and it is the responsibility of the user to label that information appropriately. TEX extracts the information in the form of TextSets, RoadRunner as sets that correspond to the instantiation of variables in the regular expressions it learns, and FiVaTech from so-called basic types in the template it learns. In spite of the differences, note that the result of the extraction process is always a collection of TextSets, which are labelled with user-defined labels in the case of SoftMealy and WIEN, but computer-generated labels in the case of TEX, RoadRunner, and FiVaTech.

Since we handcrafted annotations for every web document in our datasets, we could find which extracted TextSet was the closest to each annotation. To validate our technique, we compared each TextSet extracted to every annotation, and computed the number of true positives (tp), false negatives (fn), and false positives (fp), which allowed us to compute precision as $P = \frac{tp}{tp+fp}$, recall as $R = \frac{tp}{tp+fn}$, and the F_1-measure as $F_1 = 2 \times \frac{P*R}{P+R}$. Given an annotation, we considered that the precision and recall to extract it correspond to the extracted TextSet with the highest F_1 measure.

Recall that TEX can introduce a bias to the algorithm that searches for shared patterns. In our experiments, we used the following heuristic to find the most appropriate bias: we first set min to 1 and max to the size of the longest input document, let it be m, and we measured precision and recall; note that this does not introduce any bias, since these values allow TEX to explore every possible shared pattern. We then set $max = \lfloor \frac{m}{2} \rfloor$ and measured precision and recall again; if there were no changes, we then set $max = \lfloor \frac{m}{4} \rfloor$ and repeated the procedure until precision or recall was affected. Similarly, we explored changes to min. We experimentally found that setting $min = 2$ and $max = \lfloor 0.05m \rfloor$ was the maximum allowable bias. This resulted in a significant reduction of CPU time, without having an impact on neither precision nor recall.

Table 4 shows our results regarding effectiveness. The first few rows provide a summary in terms of mean and standard deviation of precision, recall, and F_1-measure regarding TEX and the other techniques. In average, TEX seems to outperform the other techniques in both precision and recall. The remaining rows provide the results we obtained for each web site. Note that some cells contain a dash, which indicates that the corresponding technique was not able to learn an extraction rule in 15 CPU minutes.

The drawbacks and the weak points of each technique were reflected in the previous results. RoadRunner and WIEN do not take disjunctions into account, which means that they have trouble with web documents in which different formats are used to display the same information; this is the reason why their precision and recall are relatively low and the standard deviation is relatively high. FiVaTech and RoadRunner could not learn the extraction rules for many datasets due to the complexity of their learning algorithms. SoftMealy has the problem known as early matching, which was reported by [44]: it does not consider the cases in which

Table 3
Dataset properties.

Category	ID	Url	Nr. Docs	Size	Errors	JTidy	Tokenisation
Books	S01	www.abebooks.com	30	37.65 ± 3.05	2.94 ± 0.28	0.03 ± 0.04	0.01 ± 0.01
	S02	www.awesomebooks.com	30	20.15 ± 2.42	2.16 ± 0.58	0.01 ± 0.01	0.00 ± 0.01
	S03	www.betterworldbooks.com	30	125.23 ± 11.57	2.30 ± 0.00	0.02 ± 0.01	0.00 ± 0.01
	S04	www.manybooks.net	30	26.84 ± 9.61	6.50 ± 2.31	0.01 ± 0.01	0.00 ± 0.01
	S05	www.waterstones.com	30	79.68 ± 26.22	6.46 ± 0.96	0.01 ± 0.01	0.01 ± 0.01
Cars	S06	www.autotrader.com	30	183.51 ± 17.78	13.66 ± 2.43	0.05 ± 0.01	0.01 ± 0.01
	S07	www.carmax.com	30	67.26 ± 2.74	9.57 ± 0.74	0.02 ± 0.01	0.01 ± 0.01
	S08	www.carzone.i.e.	30	71.05 ± 1.65	5.94 ± 0.33	0.01 ± 0.01	0.00 ± 0.01
	S09	www.classiccarsforsale.co.uk	30	76.02 ± 16.76	1.25 ± 0.06	0.01 ± 0.01	0.01 ± 0.01
	S10	www.internetautoguide.com	30	154.22 ± 16.35	8.20 ± 0.53	0.02 ± 0.01	0.01 ± 0.01
Events	S11	events.linkedin.com	30	9.89 ± 3.81	1.18 ± 0.24	0.00 ± 0.01	0.00 ± 0.00
	S12	www.allconferences.com	30	17.83 ± 2.27	1.52 ± 0.03	0.01 ± 0.01	0.00 ± 0.00
	S13	www.mbendi.com	30	6.95 ± 0.09	1.35 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
	S14	www.netlib.org	30	2.13 ± 0.86	0.35 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
	S15	www.rdlearning.org.uk	30	4.23 ± 0.64	0.70 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
Doctors	S16	doctor.webmd.com	30	59.23 ± 0.94	1.20 ± 0.03	0.01 ± 0.01	0.01 ± 0.01
	S17	extapps.ama-assn.org	30	24.87 ± 0.18	1.80 ± 0.00	0.01 ± 0.01	0.00 ± 0.01
	S18	www.dentists.com	30	11.92 ± 1.43	5.16 ± 2.11	0.01 ± 0.01	0.00 ± 0.00
	S19	www.drscore.com	30	23.78 ± 0.67	1.65 ± 0.82	0.00 ± 0.01	0.00 ± 0.00
	S20	www.steadyhealth.com	30	81.39 ± 0.25	1.20 ± 0.00	0.01 ± 0.01	0.00 ± 0.01
Jobs	S21	careers.insightintodiversity.com	30	30.36 ± 1.45	3.35 ± 0.45	0.01 ± 0.01	0.01 ± 0.01
	S22	www.4jobs.com	30	79.76 ± 3.57	5.52 ± 2.23	0.02 ± 0.01	0.01 ± 0.01
	S23	www.6figurejobs.com	30	72.79 ± 1.82	8.47 ± 0.06	0.02 ± 0.01	0.00 ± 0.01
	S24	www.careerbuilder.com	30	54.17 ± 3.10	4.70 ± 0.30	0.01 ± 0.01	0.01 ± 0.01
	S25	www.jobofmine.com	30	23.90 ± 2.86	2.05 ± 0.01	0.00 ± 0.01	0.00 ± 0.00
Movies	S26	www.albaniam.com	30	5.70 ± 0.10	0.60 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
	S27	www.allmovie.com	30	33.79 ± 5.24	2.97 ± 0.10	0.01 ± 0.01	0.00 ± 0.01
	S28	www.citwf.com	30	19.50 ± 0.54	1.05 ± 0.02	0.00 ± 0.01	0.00 ± 0.01
	S29	www.disneymovieslist.com	30	47.26 ± 8.84	1.62 ± 0.20	0.01 ± 0.01	0.00 ± 0.00
	S30	www.imdb.com	30	97.35 ± 3.63	6.94 ± 0.16	0.02 ± 0.01	0.01 ± 0.01
	S31	www.soulfilms.com	30	28.48 ± 7.89	3.31 ± 0.14	0.00 ± 0.01	0.00 ± 0.01
Real estate	S32	realestate.yahoo.com	30	93.94 ± 12.22	14.61 ± 0.30	0.02 ± 0.01	0.01 ± 0.01
	S33	www.haart.co.uk	30	89.64 ± 8.85	2.00 ± 0.21	0.02 ± 0.01	0.00 ± 0.01
	S34	www.homes.com	30	59.32 ± 10.07	5.00 ± 0.82	0.01 ± 0.01	0.00 ± 0.01
	S35	www.remax.com	30	69.98 ± 3.19	3.79 ± 0.14	0.01 ± 0.01	0.00 ± 0.01
	S36	www.trulia.com	30	175.39 ± 6.43	15.64 ± 0.49	0.05 ± 0.01	0.01 ± 0.01
Sports	S37	baseball.playerprofiles.com	30	20.89 ± 6.86	1.75 ± 0.18	0.00 ± 0.01	0.00 ± 0.00
	S38	en.uefa.com	30	63.42 ± 12.22	1.59 ± 0.00	0.02 ± 0.01	0.00 ± 0.01
	S39	www.atpworldtour.com	30	135.55 ± 12.29	4.60 ± 1.49	0.05 ± 0.01	0.01 ± 0.01
	S40	www.nfl.com	30	94.92 ± 1.82	4.21 ± 0.07	0.02 ± 0.01	0.01 ± 0.01
	S41	www.soccerbase.com	30	85.02 ± 21.04	7.82 ± 0.52	0.02 ± 0.01	0.01 ± 0.01
EXALG	S42	cars.amazon.com	21	25.16 ± 1.88	1.00 ± 0.00	0.00 ± 0.01	0.00 ± 0.01
	S43	players.uefa.com	20	12.09 ± 1.06	0.52 ± 0.03	0.00 ± 0.01	0.00 ± 0.00
	S44	popartist.amazon.com	19	34.17 ± 10.26	1.75 ± 0.00	0.00 ± 0.01	0.01 ± 0.01
	S45	teams.uefa.com	20	6.87 ± 0.05	1.64 ± 0.44	0.00 ± 0.01	0.00 ± 0.00
	S46	www.ausopen.com	29	41.22 ± 4.73	3.34 ± 0.03	0.01 ± 0.01	0.00 ± 0.01
	S47	www.ebay.com	50	26.43 ± 2.34	0.91 ± 0.94	0.01 ± 0.01	0.00 ± 0.01
	S48	www.majorleaguebaseball.com	9	40.10 ± 7.74	1.30 ± 0.00	0.00 ± 0.01	0.00 ± 0.01
	S49	www.netflix.com	50	43.90 ± 2.76	6.29 ± 0.49	0.01 ± 0.01	0.00 ± 0.01
	S50	www.rpmfind.net	20	34.68 ± 81.49	0.50 ± 0.02	0.00 ± 0.01	0.00 ± 0.01
RISE	S51	www.bigbook.com	235	24.73 ± 5.91	1.03 ± 0.30	0.00 ± 0.01	0.00 ± 0.01
	S52	www.iaf.net	252	14.24 ± 3.60	0.66 ± 0.21	0.00 ± 0.00	0.00 ± 0.00
	S53	okra.ucr.edu	10	7.76 ± 8.13	0.77 ± 0.82	0.00 ± 0.01	0.00 ± 0.00
	S54	www.laweekly.com/restaurants	28	5.16 ± 3.76	0.25 ± 0.14	0.00 ± 0.00	0.00 ± 0.00
	S55	www.zagat.com	91	18.23 ± 1.04	1.60 ± 0.49	0.00 ± 0.01	0.00 ± 0.00

the learnt rules match before the start or the end of the information to be extracted, which reduces its effectiveness.

4.4. Efficiency analysis

Table 5 shows our results regarding efficiency of TEX and the other techniques. The table shows the learning time (LT), extraction time per web document (ET), and the total execution time (TT) for each technique. (Note that TEX does not learn extraction rules, so we use NA in the corresponding column to mean not applicable. All timings are expressed in CPU seconds.) The first few rows provide a summary in terms of mean and standard deviation for each of the variables. The learning time and the extraction time do not include the time consumed by JTidy in the case of RoadRunner and FiVaTech, neither they include the tokenisation time in the case of TEX, SoftMealy, and WIEN. The reason is that the time to clean or tokenise a document is not actually an intrinsic feature of the proposals being analysed; these times are included in the total execution time.

In average, the extraction times per web document in the cases of RoadRunner and FiVaTech are very close to the times required by TEX, which outperforms the other techniques regarding the

Table 4
Comparison of effectiveness.

		TEX			RoadRunner			FivaTech			SoftMealy			WIEN		
		P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
	Summary															
	Mean	0.96	0.95	0.95	0.36	0.36	0.35	0.80	0.87	0.81	0.84	0.61	0.66	0.72	0.61	0.64
	StDev	0.07	0.11	0.09	0.45	0.45	0.44	0.20	0.17	0.17	0.15	0.32	0.30	0.24	0.31	0.29
	Site	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Books	S01	1.00	1.00	1.00	–	–	–	0.92	0.99	0.95	0.87	0.58	0.70	0.52	0.16	0.24
	S02	1.00	0.87	0.93	1.00	1.00	1.00	0.85	1.00	0.92	1.00	0.39	0.56	0.77	0.26	0.39
	S03	0.99	1.00	0.99	0.00	0.00	0.00	0.99	0.96	0.97	0.98	0.99	0.98	0.43	0.35	0.39
	S04	0.99	0.99	0.99	–	–	–	0.77	0.97	0.86	0.99	0.99	0.99	0.25	0.23	0.24
	S05	0.96	1.00	0.98	1.00	0.89	0.94	1.00	0.94	0.97	1.00	1.00	1.00	0.71	0.67	0.69
Cars	S06	0.99	1.00	1.00	0.00	0.00	0.00	–	–	–	0.89	0.87	0.88	0.89	0.00	0.00
	S07	1.00	1.00	1.00	0.00	0.00	0.00	0.45	0.89	0.60	0.89	0.89	0.89	0.88	0.88	0.88
	S08	0.98	1.00	0.99	0.00	0.00	0.00	0.92	1.00	0.96	0.92	0.02	0.05	0.82	0.83	0.83
	S09	0.86	0.90	0.88	0.00	0.00	0.00	–	–	–	0.90	0.90	0.90	0.17	0.13	0.15
	S10	1.00	1.00	1.00	0.00	0.00	0.00	0.97	0.94	0.96	–	–	–	0.11	0.11	0.11
Events	S11	0.96	0.96	0.96	0.74	0.74	0.74	–	–	–	0.87	0.55	0.68	0.57	0.20	0.30
	S12	0.98	0.99	0.99	–	–	–	0.84	0.90	0.87	0.99	0.25	0.40	0.80	0.40	0.53
	S13	1.00	1.00	1.00	0.90	1.00	0.95	0.90	1.00	0.95	0.60	0.60	0.60	0.80	0.40	0.53
	S14	0.96	0.98	0.97	0.00	0.00	0.00	0.39	0.50	0.44	0.87	0.44	0.59	0.40	0.40	0.40
	S15	0.99	0.99	0.99	0.00	0.00	0.00	0.99	0.79	0.88	0.52	0.39	0.45	0.62	0.23	0.34
Doctors	S16	1.00	1.00	1.00	0.00	0.00	0.00	0.77	1.00	0.87	0.86	0.45	0.59	0.60	0.60	0.60
	S17	0.98	1.00	0.99	–	–	–	–	–	–	0.79	0.39	0.53	0.60	0.60	0.60
	S18	0.92	1.00	0.96	1.00	1.00	1.00	0.56	0.99	0.72	0.61	0.61	0.61	1.00	1.00	1.00
	S19	1.00	1.00	1.00	1.00	1.00	1.00	0.78	1.00	0.88	0.91	0.87	0.89	0.78	0.80	0.79
	S20	1.00	1.00	1.00	0.00	0.00	0.00	0.83	0.83	0.83	0.75	0.25	0.38	0.75	0.75	0.75
Jobs	S21	0.83	0.83	0.83	0.70	0.70	0.70	1.00	0.74	0.85	0.57	0.47	0.52	1.00	1.00	1.00
	S22	0.92	0.98	0.95	0.00	0.00	0.00	–	–	–	0.45	0.25	0.32	0.94	0.94	0.94
	S23	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.99	0.75	0.00	0.00	0.25	0.25	0.25
	S24	1.00	1.00	1.00	0.00	0.00	0.00	0.80	0.83	0.82	0.75	0.00	0.00	0.75	0.75	0.75
	S25	0.86	1.00	0.93	0.86	1.00	0.93	–	–	–	0.75	0.03	0.06	0.50	0.50	0.50
Movies	S26	0.95	0.98	0.96	0.81	1.00	0.89	0.82	0.81	0.81	0.85	0.40	0.54	0.87	0.24	0.38
	S27	0.97	0.96	0.96	0.27	0.30	0.28	0.79	0.74	0.77	0.93	0.29	0.44	0.13	0.07	0.09
	S28	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.92	0.72	0.81	0.39	0.30	0.34
	S29	1.00	1.00	1.00	0.00	0.00	0.00	0.71	0.67	0.69	0.97	0.97	0.97	0.72	0.72	0.72
	S30	0.93	0.86	0.89	0.00	0.00	0.00	–	–	–	0.88	0.85	0.86	0.38	0.38	0.38
	S31	0.99	0.92	0.95	0.00	0.00	0.00	0.59	1.00	0.74	0.99	0.95	0.97	0.91	0.81	0.86
Real estate	S32	1.00	1.00	1.00	0.00	0.00	0.00	0.77	0.97	0.86	1.00	1.00	1.00	0.83	0.83	0.83
	S33	1.00	1.00	1.00	0.00	0.00	0.00	0.94	1.00	0.97	1.00	1.00	1.00	0.78	0.75	0.76
	S34	0.99	0.99	0.99	0.00	0.00	0.00	–	–	–	0.80	0.79	0.79	0.92	0.92	0.92
	S35	0.70	0.98	0.82	–	–	–	0.77	0.99	0.87	0.84	0.84	0.84	1.00	1.00	1.00
	S36	0.63	1.00	0.77	0.00	0.00	0.00	–	–	–	0.88	0.92	0.90	1.00	0.89	0.94
Sports	S37	1.00	1.00	1.00	0.00	0.00	0.00	0.36	0.99	0.52	0.83	0.13	0.23	1.00	1.00	1.00
	S38	1.00	1.00	1.00	0.00	0.00	0.00	–	–	–	1.00	1.00	1.00	1.00	1.00	1.00
	S39	0.97	0.99	0.98	0.00	0.00	0.00	0.99	0.88	0.93	0.94	0.94	0.94	0.71	0.71	0.71
	S40	1.00	1.00	1.00	0.93	1.00	0.97	0.53	0.81	0.64	0.71	0.71	0.71	0.86	0.86	0.86
	S41	0.97	1.00	0.98	0.00	0.00	0.00	–	–	–	0.89	0.89	0.89	0.89	0.89	0.89
EXALG	S42	0.93	0.73	0.82	0.27	0.33	0.30	0.60	0.67	0.63	0.98	1.00	0.99	0.97	1.00	0.98
	S43	1.00	0.90	0.95	0.92	0.92	0.92	0.91	0.94	0.92	0.92	0.90	0.91	0.92	0.51	0.66
	S44	1.00	0.98	0.99	0.99	0.99	0.99	1.00	1.00	1.00	0.94	0.72	0.82	0.92	0.58	0.71
	S45	0.99	0.99	0.99	0.90	0.92	0.91	0.97	0.99	0.98	0.81	0.89	0.85	0.64	0.75	0.69
	S46	1.00	1.00	1.00	0.37	0.39	0.38	0.24	0.82	0.37	0.65	0.28	0.39	0.67	0.32	0.43
	S47	0.97	1.00	0.98	0.00	0.00	0.00	0.83	1.00	0.91	0.70	0.12	0.20	0.70	0.12	0.20
	S48	0.98	0.55	0.70	0.00	0.00	0.00	0.99	1.00	0.99	0.99	0.46	0.63	0.65	0.33	0.44
	S49	0.99	0.99	0.99	–	–	–	0.82	0.80	0.81	0.94	0.82	0.88	0.99	0.99	0.99
	S50	0.95	0.97	0.96	0.98	0.99	0.98	0.99	0.41	0.58	0.72	0.03	0.06	0.99	0.99	0.99
RISE	S51	0.95	0.94	0.94	1.00	1.00	1.00	–	–	–	0.81	0.77	0.79	0.68	0.98	0.80
	S52	0.84	0.38	0.52	0.00	0.00	0.00	0.53	0.69	0.60	0.37	0.43	0.40	1.00	1.00	1.00
	S53	1.00	0.82	0.90	0.96	0.56	0.71	0.49	0.34	0.40	0.83	0.82	0.82	0.60	0.67	0.63
	S54	0.97	0.92	0.94	0.00	0.00	0.00	0.83	0.57	0.68	0.87	0.49	0.63	0.90	0.80	0.85
	S55	1.00	0.86	0.92	0.00	0.00	0.00	1.00	0.98	0.99	0.81	0.63	0.71	0.81	0.72	0.76

total execution time. (In the table, 0.00 ± 0.00 means that the time was less than one millisecond, but the resolution of our timer was not enough to measure such small amounts of time.)

The drawbacks of the other techniques were also reflected in their efficiency results. RoadRunner relies on a backtracking algorithm that may need too much time to learn an extraction rule and its matching algorithm has exponential time complexity with respect to the input document size in the worst case; FiVaTech relies on a clustering algorithm that builds on a tree-edit distance that is time consuming. These are the reasons why RoadRunner and FiVaTech failed to learn an extraction rule in 6 and 14 cases, respectively, after consuming 15 CPU minutes. SoftMealy needs to match a couple of regular expressions before it can extract an attribute, and there can be multiple couples of regular expressions

Table 5
Comparison of efficiency.

		TEX			RoadRunner			FiVaTech			SoftMealy			WIEN		
		LT	ET	TT	LT	ET	TT	LT	ET	TT	LT	ET	TT	LT	ET	TT
	Summary															
	Mean	NA	0.01	0.81	20.03	0.00	20.41	122.94	0.01	123.48	7.10	0.79	53.63	7.80	0.66	18.99
	StDev	NA	0.02	0.86	123.71	0.00	123.68	196.42	0.01	196.60	5.13	0.21	57.61	5.15	0.27	14.82
	Site	LT	ET	TT	LT	ET	TT	LT	ET	TT	LT	ET	TT	LT	ET	TT
Books	S01	NA	0.00 ± 0.00	0.44	–	–	–	15.46	0.00 ± 0.01	16.50	9.09	0.87 ± 0.23	36.33	9.66	0.52 ± 0.48	20.20
	S02	NA	0.00 ± 0.00	0.28	0.92	0.00 ± 0.00	1.17	8.14	0.00 ± 0.01	8.53	5.68	1.00 ± 0.00	22.39	5.01	0.77 ± 0.38	11.47
	S03	NA	0.03 ± 0.00	1.48	0.98	0.00 ± 0.00	1.62	85.32	0.01 ± 0.01	86.35	16.15	0.98 ± 0.03	58.00	15.57	0.43 ± 0.48	33.07
	S04	NA	0.02 ± 0.00	0.62	–	–	–	65.49	0.00 ± 0.01	65.80	7.38	0.99 ± 0.01	37.80	6.74	0.25 ± 0.53	14.98
	S05	NA	0.01 ± 0.00	0.59	1.14	0.00 ± 0.00	1.61	51.53	0.07 ± 0.05	53.96	8.67	1.00 ± 0.00	86.35	8.02	0.71 ± 0.38	17.05
Cars	S06	NA	0.04 ± 0.00	1.98	0.83	0.00 ± 0.00	2.15	–	–	–	14.87	0.89 ± 0.26	119.34	14.18	0.89 ± 0.27	28.86
	S07	NA	0.03 ± 0.00	1.11	1.72	0.00 ± 0.00	2.29	34.21	0.01 ± 0.01	34.98	7.66	0.89 ± 0.23	30.84	7.43	0.88 ± 0.27	15.33
	S08	NA	0.01 ± 0.00	0.30	0.69	0.00 ± 0.00	1.08	446.90	0.02 ± 0.01	447.88	5.30	0.92 ± 0.27	34.27	4.80	0.82 ± 0.36	9.66
	S09	NA	0.02 ± 0.00	0.86	1.47	0.00 ± 0.00	1.78	–	–	–	10.92	0.90 ± 0.27	89.81	10.03	0.17 ± 0.51	20.65
	S10	NA	0.02 ± 0.00	0.87	4.74	0.00 ± 0.00	5.49	117.16	0.01 ± 0.01	118.09	8.74	–	–	7.78	0.11 ± 0.51	17.14
Events	S11	NA	0.01 ± 0.00	0.16	2.57	0.00 ± 0.00	2.65	–	–	–	5.04	0.87 ± 0.30	47.08	4.15	0.57 ± 0.52	7.49
	S12	NA	0.01 ± 0.00	0.42	–	–	–	42.96	0.00 ± 0.01	43.18	10.06	0.99 ± 0.01	72.40	7.27	0.80 ± 0.45	12.11
	S13	NA	0.00 ± 0.00	0.08	0.45	0.00 ± 0.00	0.51	1.56	0.00 ± 0.00	1.64	2.61	0.60 ± 0.55	8.38	1.86	0.80 ± 0.45	3.17
	S14	NA	0.00 ± 0.00	0.05	0.17	0.00 ± 0.00	0.17	0.27	0.00 ± 0.00	0.30	6.65	0.87 ± 0.23	14.54	4.43	0.40 ± 0.55	6.77
	S15	NA	0.00 ± 0.00	0.06	0.45	0.00 ± 0.00	0.48	6.21	0.00 ± 0.00	6.26	4.62	0.52 ± 0.42	11.75	3.24	0.62 ± 0.49	5.23
Doctors	S16	NA	0.00 ± 0.00	0.34	0.73	0.00 ± 0.00	1.03	11.81	0.00 ± 0.00	12.14	8.28	0.86 ± 0.15	38.50	9.45	0.60 ± 0.52	24.27
	S17	NA	0.00 ± 0.00	0.30	–	–	–	–	–	–	6.07	0.79 ± 0.41	16.82	6.99	0.60 ± 0.52	14.59
	S18	NA	0.00 ± 0.00	0.11	867.63	0.00 ± 0.00	867.77	9.94	0.00 ± 0.01	10.16	2.28	0.61 ± 0.50	10.50	1.97	1.00 ± 0.00	3.87
	S19	NA	0.00 ± 0.00	0.20	3.28	0.00 ± 0.00	3.35	25.35	0.00 ± 0.01	25.47	4.49	0.91 ± 0.19	22.06	3.88	0.78 ± 0.40	7.57
	S20	NA	0.10 ± 0.00	3.21	0.67	0.00 ± 0.00	1.01	9.59	0.00 ± 0.01	10.03	9.70	0.75 ± 0.41	49.89	9.56	0.75 ± 0.41	19.52
Jobs	S21	NA	0.00 ± 0.00	0.33	0.78	0.00 ± 0.00	1.03	13.76	0.00 ± 0.01	14.10	7.77	0.57 ± 0.50	24.68	7.49	1.00 ± 0.00	14.82
	S22	NA	0.01 ± 0.00	0.89	0.69	0.00 ± 0.00	1.33	–	–	–	9.09	0.45 ± 0.53	46.19	9.02	0.94 ± 0.10	19.02
	S23	NA	0.01 ± 0.00	0.72	1.83	0.00 ± 0.00	2.40	90.78	0.01 ± 0.01	91.67	11.76	0.75 ± 0.50	39.08	11.56	0.25 ± 0.50	24.02
	S24	NA	0.01 ± 0.00	0.39	0.53	0.00 ± 0.00	0.89	266.00	0.01 ± 0.01	266.67	8.13	0.75 ± 0.50	59.06	8.11	0.75 ± 0.50	16.05
	S25	NA	0.01 ± 0.00	0.45	1.86	0.00 ± 0.00	1.98	–	–	–	5.19	0.75 ± 0.50	27.85	5.09	0.50 ± 0.58	9.95
Movies	S26	NA	0.00 ± 0.00	0.17	0.64	0.00 ± 0.00	0.69	1.59	0.00 ± 0.00	1.62	3.65	0.85 ± 0.40	7.16	2.70	0.87 ± 0.41	4.43
	S27	NA	0.03 ± 0.00	1.00	1.64	0.00 ± 0.00	1.95	14.84	0.00 ± 0.01	15.23	11.29	0.93 ± 0.41	50.34	7.78	0.13 ± 0.52	13.88
	S28	NA	0.00 ± 0.00	0.12	0.69	0.00 ± 0.00	0.80	29.70	0.00 ± 0.00	29.84	3.70	0.92 ± 0.41	14.12	2.79	0.39 ± 0.41	6.61
	S29	NA	0.01 ± 0.00	0.53	0.59	0.00 ± 0.00	0.76	259.23	0.00 ± 0.01	259.48	4.34	0.97 ± 0.03	49.28	3.95	0.72 ± 0.40	8.02
	S30	NA	0.06 ± 0.00	2.54	0.97	0.00 ± 0.00	1.45	–	–	–	16.38	0.88 ± 0.38	80.22	15.87	0.38 ± 0.54	32.92
	S31	NA	0.00 ± 0.00	0.34	0.47	0.00 ± 0.00	0.67	17.24	0.00 ± 0.00	17.46	10.64	0.99 ± 0.03	48.41	9.73	0.91 ± 0.41	19.28
Real estate	S32	NA	0.02 ± 0.00	1.26	3.10	0.00 ± 0.00	3.62	246.95	0.03 ± 0.02	248.35	16.32	1.00 ± 0.00	103.49	15.62	0.83 ± 0.41	32.99
	S33	NA	0.01 ± 0.00	0.48	2.75	0.00 ± 0.00	3.31	20.76	0.00 ± 0.01	21.40	7.43	1.00 ± 0.00	108.50	7.04	0.78 ± 0.37	14.21
	S34	NA	0.02 ± 0.00	0.56	1.39	0.00 ± 0.00	1.78	–	–	–	8.25	0.80 ± 0.39	74.23	8.22	0.92 ± 0.19	16.49
	S35	NA	0.04 ± 0.00	1.45	–	–	–	–	–	–	7.60	0.84 ± 0.38	33.59	7.52	1.00 ± 0.00	15.35
	S36	NA	0.12 ± 0.00	4.40	2.18	0.00 ± 0.00	3.56	–	–	–	25.55	0.88 ± 0.26	311.36	25.18	1.00 ± 0.00	51.42
Sports	S37	NA	0.02 ± 0.00	0.66	1.37	0.00 ± 0.00	1.48	14.68	0.00 ± 0.01	14.84	5.51	0.83 ± 0.32	23.12	5.34	1.00 ± 0.00	10.48
	S38	NA	0.01 ± 0.00	0.53	0.64	0.00 ± 0.00	1.05	–	–	–	5.99	1.00 ± 0.00	39.50	5.83	1.00 ± 0.00	11.82
	S39	NA	0.06 ± 0.00	2.29	1.11	0.00 ± 0.00	2.68	111.03	0.03 ± 0.03	113.41	19.14	0.94 ± 0.13	202.05	18.75	0.71 ± 0.42	37.61
	S40	NA	0.02 ± 0.00	1.09	1.65	0.00 ± 0.00	2.28	159.39	0.01 ± 0.01	160.40	10.14	0.71 ± 0.42	50.51	9.64	0.86 ± 0.32	19.67
	S41	NA	0.04 ± 0.00	1.72	2.01	0.00 ± 0.00	2.48	–	–	–	11.54	0.89 ± 0.32	59.84	11.04	0.89 ± 0.32	22.42
EXALG	S42	NA	0.00 ± 0.00	0.08	0.55	0.00 ± 0.00	0.62	2.29	0.00 ± 0.01	2.40	1.06	0.50 ± 0.36	11.19	19.47	1.00 ± 0.00	27.60
	S43	NA	0.01 ± 0.00	0.16	0.51	0.00 ± 0.00	0.61	10.81	0.00 ± 0.01	10.94	1.34	0.46 ± 0.39	13.01	3.53	0.72 ± 0.51	7.25
	S44	NA	0.00 ± 0.00	0.23	0.98	0.00 ± 0.00	1.06	246.97	0.01 ± 0.01	247.12	3.42	0.27 ± 0.55	26.96	15.66	0.32 ± 0.51	26.07
	S45	NA	0.01 ± 0.00	0.16	0.28	0.00 ± 0.00	0.33	0.89	0.00 ± 0.00	0.94	1.00	0.99 ± 0.03	4.32	1.79	0.21 ± 0.29	4.07

	S46	NA	0.01 ± 0.00	0.70	1.15	0.00 ± 0.00	1.44	132.24	0.01 ± 0.01	132.77	1.98	0.98 ± 0.06	24.32	9.67	0.71 ± 0.49	26.91
	S47	NA	0.04 ± 0.00	2.46	2.51	0.00 ± 0.00	2.82	577.97	0.01 ± 0.01	578.75	2.23	0.75 ± 0.50	18.49	5.44	0.75 ± 0.50	25.21
	S48	NA	0.11 ± 0.00	1.12	0.95	0.00 ± 0.01	1.01	158.33	0.01 ± 0.01	158.43	1.29	0.15 ± 0.04	21.06	3.60	0.12 ± 0.08	9.89
	S49	NA	0.01 ± 0.00	1.00	–	–	–	706.64	0.02 ± 0.01	707.81	3.68	0.38 ± 0.52	38.14	7.47	0.80 ± 0.45	39.33
	S50	NA	0.00 ± 0.00	0.37	1.31	0.00 ± 0.00	1.42	–	–	–	1.40	0.80 ± 0.05	27.88	1.29	0.75 ± 0.50	11.97
RISE	S51	NA	0.00 ± 0.00	1.39	12.04	0.00 ± 0.00	13.28	–	–	–	1.50	0.29 ± 0.37	168.89	2.68	0.19 ± 0.43	92.98
	S52	NA	0.00 ± 0.00	0.23	0.89	0.00 ± 0.00	0.92	14.41	0.01 ± 0.01	14.49	1.39	0.54 ± 0.25	14.76	2.00	0.75 ± 0.45	4.43
	S53	NA	0.02 ± 0.00	0.66	11.23	0.00 ± 0.00	11.84	849.27	0.00 ± 0.01	850.17	1.73	0.84 ± 0.21	218.53	1.64	0.72 ± 0.34	37.16
	S54	NA	0.00 ± 0.00	0.23	0.37	0.00 ± 0.00	0.41	4.24	0.00 ± 0.00	4.32	2.18	0.74 ± 0.13	25.82	0.90	0.41 ± 0.47	3.82
	S55	NA	0.00 ± 0.00	0.14	33.60	0.00 ± 0.00	33.77	158.48	0.00 ± 0.00	158.70	2.57	0.85 ± 0.34	22.99	13.60	0.78 ± 0.50	33.38

Table 6
Statistical ranking.

Ranking variable	Friedman's p-value	Bergmann's ranking	
		Technique	Rank
P	5.96475E−09	TEX	1.61
		FiVaTech	3.04
		SoftMealy	3.04
		WIEN	3.35
		RoadRunner	3.96
R	1.40668E−08	TEX	1.80
		FiVaTech	2.43
		SoftMealy	3.45
		WIEN	3.57
		RoadRunner	3.76
F_1	2.09518E−09	TEX	1.55
		FiVaTech	2.85
		SoftMealy	3.27
		WIEN	3.47
		RoadRunner	3.85
TT	9.9681E−11	TEX	0.20
		RoadRunner	0.26
		WIEN	0.38
		FiVaTech	0.55
		SoftMealy	0.61

since it builds on non-deterministic transducers; WIEN requires to match a prefix and a suffix regular expressions for every attribute to be extracted. They both have to search for the delimiters that match the rules learnt, which may require to perform several passes through the input documents, thus increasing their extraction time per web document significantly compared to other techniques. Note that SoftMealy learnt a rule for site $S10$ in 8.74 seconds, but its extraction algorithm did not finish and returned an out of memory error.

4.5. Statistical ranking

In order to confirm our conclusions, we used statistical inference at the standard confidence level $\alpha = 0.05$, cf. Table 6. We first used Friedman's test [19] to check if P, R, F_1, and TT could be considered statistically equal or not for each technique; note that Friedman's p-value is largely less than α in every case, which is a strong indication that the precisions, recalls, and the total execution times cannot be considered statistically equal. We then used Bergmann's test [19] to rank the techniques and found out that there is a strong statistical evidence that TEX outperforms RoadRunner, FiVaTech, SoftMealy, and WIEN in terms of precision, recall, and, consequently, F_1-measure. TEX also outperforms the other techniques in terms of total execution time.

5. Conclusions

In this article, we have presented an information extraction approach called TEX. It is based on the idea that web documents that are generated by the same server side template share tokens, and that these tokens contain irrelevant information since they are parts of the server-side template that was used to generate them.

TEX is a completely unsupervised information extractor that saves end users from the burden of annotating training examples to learn extraction rules, and from maintaining extraction rules. TEX allows working on malformed web documents since it does not require converting HTML code into XHTML or to build DOM trees, which reduces its extraction time. Furthermore, it does not need the information in the input web documents to be formatted using repetitive patterns.

We have studied the complexity of TEX and demonstrated that it is computationally tractable. Our empirical analysis of TEX on a collection of real-world datasets has proved that our technique achieves a very high precision and recall, which are very close to 100%. Our comparison and statistical analysis has shown that our technique performs better than other techniques in the literature.

Acknowledgements

We greatly appreciate the help of our reviewers to improve the contents of the article and to suggest new research directions. Our work was supported by the European Commission (FEDER), the Spanish and the Andalusian R&D&I Programmes (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] Manuel Álvarez, Alberto Pan, Juan Raposo, Fernando Bellas, Fidel Cacheda, Extracting lists of data records from semi-structured web pages, Data Knowl. Eng. 64 (2) (2008) 491–509.
[2] Arvind Arasu, Hector Garcia-Molina, Extracting structured data from web pages, in: SIGMOD Conference, 2003, pp. 337–348.
[3] José Luis Arjona, Rafael Corchuelo, David Ruiz, Miguel Toro, From wrapping to knowledge, IEEE Trans. Knowl. Data Eng. 19 (2) (2007) 310–323.
[4] Gustavo O. Arocena, Alberto O. Mendelzon, WebOQL: restructuring documents, databases, and webs, Theory Pract. Object Syst. 5 (3) (1999) 127–141.
[5] Fatima Ashraf, Tansel Özyer, Reda Alhajj, Employing clustering techniques for automatic information extraction from HTML documents, IEEE Trans. Syst. Man Cybernet. Part C 38 (5) (2008) 660–673.
[6] David Buttler, Ling Liu, Calton Pu, A fully automated object extraction system for the World Wide Web, in: ICDCS, 2001, pp. 361–370.
[7] Deng Cai, Shipeng Yu, Ji-RongWen, Wei-Ying Ma, Extracting content structure for web pages based on visual representation, in: Asia-Pacific Web Conference, 2003, pp. 406–417.
[8] Mary Elaine Califf, Raymond J. Mooney, Bottom-up relational learning of pattern matching rules for information extraction, J. Mach. Learn. Res. 4 (2003) 177–210.
[9] Andrew Carlson, Charles Schafer, Bootstrapping information extraction from semi-structured web pages, in: ECML/PKDD (1), 2008, pp. 195–210.
[10] Chia-Hui Chang, Mohammed Kayed, Moheb R. Girgis, Khaled F. Shaalan, A survey of web information extraction systems, IEEE Trans. Knowl. Data Eng. 18 (10) (2006) 1411–1428.
[11] Chia-Hui Chang, Shih-Chien Kuo, OLERA: semisupervised web-data extraction with visual support, IEEE Intell. Syst. 19 (6) (2004) 56–64.
[12] Chia-Hui Chang, Shao-Chen Lui, IEPAD: information extraction based on pattern discovery, in: WWW, 2001, pp. 681–688.
[13] Boris Chidlovskii, Bruno Roustant, Marc Brette, Documentum ECI self-repairing wrappers: performance analysis, in: SIGMOD Conference, 2006, pp. 708–717.
[14] William W. Cohen, Matthew Hurst, Lee S. Jensen, A flexible learning system for wrapping tables and lists in HTML documents, in: WWW, 2002, pp. 232–241.
[15] Valter Crescenzi, Giansalvatore Mecca, Grammars have exceptions, Inf. Syst. 23 (8) (1998) 539–565.
[16] Valter Crescenzi, Giansalvatore Mecca, Automatic information extraction from large websites, J. ACM 51 (5) (2004) 731–779.
[17] Hazem Elmeleegy, Jayant Madhavan, Alon Y. Halevy, Harvesting relational tables from lists on the Web, PVLDB 2 (1) (2009) 1078–1089.
[18] Dayne Freitag, Information extraction from HTML: application of a general machine learning approach, in: AAAI, 1998, pp. 517–523.
[19] Salvador Garcfa, Francisco Herrera, Extension on statistical comparisons of classifiers over multiple data sets for all pairwise comparisons, J. Mach. Learn. Res. 9 (2008) 2677–2694.
[20] Dawn G. Gregg, Steven Walczak, Exploiting the Information Web, IEEE Trans. Syst. Man Cybernet. Part C 37 (1) (2007) 109–125.
[21] Pankaj Gulhane, Amit Madaan, Rupesh R. Mehta, Jeyashankher Ramamirtham, Rajeev Rastogi, Sandeepkumar Satpal, Srinivasan H. Sengamedu, Ashwin Tengli, Charu Tiwari, Web-scale information extraction with Vertex, in: ICDE, 2011, pp. 1209–1220.
[22] Pankaj Gulhane, Rajeev Rastogi, Srinivasan H. Sengamedu, Ashwin Tengli, Exploiting content redundancy for web information extraction, in: WWW, 2010, pp. 1105–1106.
[23] Rahul Gupta, Sunita Sarawagi, Answering table augmentation queries from unstructured lists on the web, PVLDB 2 (1) (2009) 289–300.
[24] Joachim Hammer, Jason McHugh, Hector Garcia-Molina, Semistructured data: the TSIMMIS experience, in: Advances in Databases and Information Systems, 1997, pp. 1–8.
[25] Jer Lang Hong, Eu-Gene Siew, Simon Egerton, Information extraction for search engines using fast heuristic techniques, Data Knowl. Eng. 69 (2) (2010) 169–196.
[26] Chun-Nan Hsu, Ming-Tzung Dung, Generating finite-state transducers for semi-structured data extraction from the Web, Inf. Syst. 23 (8) (1998) 521–538.
[27] Jinbeom Kang, Joongmin Choi, Recognising informative web page blocks using visual segmentation for efficient information extraction, J. UCS 14 (11) (2008) 1893–1910.
[28] Kayed Mohammed, Chang Chia-Hui, FiVaTech: page-level web data extraction from template pages, IEEE Trans. Knowl. Data Eng. (2010).
[29] Donald E. Knuth, James H. Morris Jr., Vaughan R. Pratt, Fast pattern matching in strings, SIAM J. Comput. 6 (2) (1977) 323–350.
[30] Raymond Kosala, Hendrik Blockeel, Maurice Bruynooghe, Jan Van den Bussche, Information extraction from structured documents using k-testable tree automaton inference, Data Knowl. Eng. 58 (2) (2006) 129–158.
[31] Nicholas Kushmerick, Regression testing for wrapper maintenance, in: AAAI/IAAI, 1999, pp. 74–79.
[32] Nicholas Kushmerick, Daniel S. Weld, Robert B. Doorenbos, Wrapper induction for information extraction, in: IJCAI (1), 1997, pp. 729–737.
[33] Kristina Lerman, Steven Minton, Craig A. Knoblock, Wrapper maintenance: a machine learning approach, J. Artif. Intell. Res. 18 (2003) 149–181.
[34] Longzhuang Li, Yonghuai Liu, Abel Obregon, Matt Weatherston, Visual segmentation-based data record extraction from web documents, in: IRI, 2007, pp. 502–507.
[35] Qingzhong Li, Yanhui Ding, An Feng, Yongquan Dong, A novel method for extracting information from web pages with multiple presentation templates, JSW, 2010.
[36] Bing Liu, Robert L. Grossman, Yanhong Zhai, Mining data records in web pages, in: KDD, 2003, pp. 601–606.
[37] Bing Liu, Robert L. Grossman, Yanhong Zhai, Mining web pages for data records, IEEE Intell. Syst. 19 (6) (2004) 49–55.
[38] Bing Liu, Yanhong Zhai, Net: a system for extracting web data from flat and nested data records, in: WISE, 2005, pp. 487–495.
[39] Wei Liu, Xiaofeng Meng, Weiyi Meng, ViDE: a vision-based approach for deep web data extraction, IEEE Trans. Knowl. Data Eng. (2010) 447–460.
[40] Ashwin Machanavajjhala, Arun Shankar Iyer, Philip Bohannon, Srujana Merugu, Collective extraction from heterogeneous web lists, in: WSDM, 2011, pp. 445–454.
[41] Xiaofeng Meng, Dongdong Hu, Chen Li, Schema-guided wrapper maintenance for web-data extraction, in: WIDM, 2003, pp. 1–8.
[42] Robert C. Miller, Brad A. Myers, Lightweight structured text processing, in: USENIX Annual Technical Conference, General Track, 1999, pp. 131–144.
[43] Ion Muslea, RISE: Repository of Online Information Sources Used in Information Extraction Tasks, 2004.
[44] Ion Muslea, Steven Minton, Craig A. Knoblock, Hierarchical wrapper induction for semistructured information sources, Autonom. Agents Multi-Agent Syst. 4 (1/2) (2001) 93–114.
[45] Nikolaos Papadakis, Dimitrios Skoutas, Konstantinos Raftopoulos, Theodora A. Varvarigou, STAVIES: a system for information extraction from unknown web data sources through automatic web wrapper generation using clustering techniques, IEEE Trans. Knowl. Data Eng. 17 (12) (2005) 1638–1652.
[46] Justin Park, Denilson Barbosa, Adaptive record extraction from web pages, in: WWW, 2007, pp. 1335–1336.
[47] David Raggett, Clean up your web pages with HP's HTML tidy, Comput. Networks 30 (1–7) (1998) 730–732.
[48] Juan Raposo, Alberto Pan, Manuel Álvarez, Ángel Viña, Automatic wrapper maintenance for semi-structured web sources using results from previous queries, in: SAC, 2005a, pp. 654–659.
[49] Juan Raposo, Alberto Pan, Manuel Álvarez, Justo Hidalgo, Automatically generating labeled examples for web wrapper maintenance, in: Web Intelligence, 2005b, pp. 250–256.
[50] Juan Raposo, Alberto Pan, Manuel Álvarez, Justo Hidalgo, Ángel Viña, The WARGO system: semi-automatic wrapper generation in presence of complex data access modes, in: DEXA Workshops, 2002, pp. 313–320.
[51] Arnaud Sahuguet, Fabien Azavant, Building intelligent web applications using lightweight wrappers, Data Knowl. Eng. 36 (3) (2001) 283–316.
[52] Yuan Kui Shen, David R. Karger, U-REST: an unsupervised record extraction system, in: WWW, 2007, pp. 1347–1348.
[53] Kai Simon, Georg Lausen, ViPER: augmenting automatic information extraction with visual perceptions, in: CIKM, 2005, pp. 381–388.
[54] Hassan A. Sleiman, Rafael Corchuelo, A reference architecture to devise web information extractors, in: CAiSE Workshops, 2012, pp. 235–248.
[55] Hassan A. Sleiman, Rafael Corchuelo, A survey on region extractors from web documents, IEEE Trans. Knowl. Data Eng. (2012) 99.
[56] Hassan A. Sleiman, Rafael Corchuelo, Towards a method for unsupervised web information extraction, in: ICWE, 2012, pp. 427–430.
[57] Hassan A. Sleiman, Rafael Corchuelo, An unsupervised technique to extract information from semi-structured web pages, in: WISE, 2012, pp.631–637.
[58] Stephen Soderland, Learning information extraction rules for semi-structured and free text, Mach. Learn. 34 (1–3) (1999) 233–272.
[59] Weifeng Su, Jiying Wang, Frederick H. Lochovsky, ODE: ontology-assisted data extraction, ACM Trans. Database Syst. 34 (2) (2009).
[60] Cui Tao, David W. Embley, Automatic hidden-web table interpretation, conceptualization, and semantic annotation, Data Knowl. Eng. 68 (7) (2009) 683–703.

[61] Jiying Wang, Frederick H. Lochovsky, Data-rich section extraction from HTML pages, in: WISE, 2002, pp. 313–322.
[62] Jiying Wang, Frederick H. Lochovsky, Data extraction and label assignment for web databases, in: WWW, 2003, pp. 187–196.
[63] Liu Wei, Xiaofeng Meng, Weiyi Meng, Vision-based web data records extraction, in: International Workshop on the Web and Databases, 2006.
[64] Yanhong Zhai, Bing Liu, Structured data extraction from the Web based on partial tree alignment, IEEE Trans. Knowl. Data Eng. 18 (12) (2006) 1614–1628.
[65] Hongkun Zhao, Weiyi Meng, Zonghuan Wu, Vijay Raghavan, Clement T. Yu, Fully automatic wrapper generation for search engines, in: WWW, 2005, pp. 66–75.
[66] Hongkun Zhao, Weiyi Meng, Clement T. Yu, Automatic extraction of dynamic record sections from search engine result pages, in: VLDB, 2006, pp. 989–1000.
[67] Jun Zhu, Zaiqing Nie, Ji-Rong Wen, Bo Zhang, Wei-Ying Ma, Simultaneous record detection and attribute labeling in web data extraction, in: KDD, 2006, pp. 494–503.