The Smallest Extraction Problem

Valerio Cetorelli
Università Roma Tre
Rome, Italy
valerio.cetorelli@uniroma3.it

Paolo Atzeni
Università Roma Tre
Rome, Italy
paolo.atzeni@uniroma3.it

Valter Crescenzi
Università Roma Tre
Rome, Italy
valter.crescenzi@uniroma3.it

Franco Milicchio
Università Roma Tre
Rome, Italy
franco.milicchio@uniroma3.it

ABSTRACT

We introduce *landmark grammars*, a new family of context-free grammars aimed at describing the HTML source code of pages published by large and templated websites and therefore at effectively tackling Web data extraction problems. Indeed, they address the inherent *ambiguity* of HTML, one of the main challenges of Web data extraction, which, despite over twenty years of research, has been largely neglected by the approaches presented in literature.

We then formalize the *Smallest Extraction Problem* (SEP), an optimization problem for finding the grammar of a family that best describes a set of pages and contextually extract their data.

Finally, we present an unsupervised learning algorithm to induce a landmark grammar from a set of pages sharing a common HTML template, and we present an automatic Web data extraction system. The experiments on consolidated benchmarks show that the approach can substantially contribute to improve the state-of-the-art.

PVLDB Reference Format:
Valerio Cetorelli, Paolo Atzeni, Valter Crescenzi, and Franco Milicchio. The Smallest Extraction Problem. PVLDB, 14(11): 2445-2458, 2021.
doi:10.14778/3476249.3476293

1 INTRODUCTION

As the Web is the largest knowledge base ever built by humans, Web data extraction has gained the attention of researchers for more than twenty years, with a number of proposals for the development of *wrappers*, that is, software programs that extract data from websites. Specific vertical solutions for the extraction of data from the Web are frequently developed and maintained by companies of every size and sector within the IT industry, and several companies vertically specialized in data extraction tasks have been founded in the last years [32, 43, 44, 47]. Despite all these efforts, the effective extraction of data at Web scale is still a challenging problem far from being effectively solved, a problem that is getting even more exacerbated as the amount of information available online keeps on growing and being updated at an ever increasing pace. Moreover, the scale of the Web, in terms of both volume and variety, implies the need for finding unsupervised approaches to wrapper construction, as human intervention would not be feasible.

It is important to observe that the vast majority of Web content is currently organized and exposed in HTML format. Indeed, despite two decades of promises, it is now apparent and widely accepted that most websites will never publicly expose their data in regularly structured formats (e.g., XML and JSON) nor by means of easy-to-consume processes (e.g., APIs). This is due to several reasons, including: data are an important asset for competitiveness and most organizations do not want to share them; second, publishing data and maintaining the publishing process is costly and most organizations do not see a tangible return on investment; third, publishing data can expose to legal issues, such as those related to the General Data Protection Regulations in Europe.

Such a widespread use of HTML (instead of XML, JSON, or APIs) makes data extraction difficult. Indeed, the main goal of HTML is to support the logical presentation of contents rather than the structuring and organization of data. This is done by means of a quite small, predefined set of element names for tags, which end up being only a loose trace of the structure [20]. As a consequence, the very same HTML tags end up being used over and over again, with different *roles*, to mark unrelated pieces of information that are presented in similar ways.

Most data on the Web are published by *templated* sites [24, 31], where scripts execute queries on underlying databases to get the data and then produce pages by embedding these data into predefined HTML templates. Even though the underlying data are often organized according to a structured (for example relational) schema, they end up being available only as HTML pages meant to be manually browsed and consumed by humans rather than being electronically processed. The original organization of data is lost in a sequence of HTML tags that have only a loose, and quite often misleading, trace of the original structure.

A major challenge that makes Web data extraction hard is that of *HTML ambiguity*. Let us comment by means of an example, on the basis of the pages sketched in Figures 1a and 1b from a fictional templated website publishing data about movies:[1] here the `<B>` and `</B>` tags are both used as part of the template (the blue colored occurrences in both pages) and as content (the red occurrence in the second page). The same observation can be applied to texts, as well: the `Price` word is used both as part of the template, i.e., a label, to identify the price of a movie (blue occurrences in both pages)

Proceedings of the VLDB Endowment, Vol. 14, No. 11 ISSN 2150-8097.
doi:10.14778/3476249.3476293

[1] We omit, for the sake of brevity, `<HTML>`, `<HEAD>` or `<BODY>` as they (should) occur in every HTML page.

<B>Title</B>
<DIV>Price</DIV>
<SPAN>Descr.</SPAN>
<DIV>The Price of Everything</DIV>

<SPAN>35.22</SPAN>
A US documentary

(a) Page a

<B>Title</B>
<DIV>Price</DIV>
<SPAN>Descr.</SPAN>
<DIV>Star Wars</DIV>

<SPAN>9.98</SPAN>
The saga directed by
<B>G. Lucas</B>

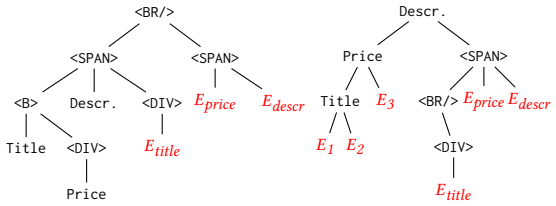
(b) Page b

<B>Title</B>
<SPAN>Descr.</SPAN>
<DIV>E_{title}</DIV>

<SPAN>E_{price}</SPAN>
E_{descr}

(c) HTML Template

Figure 1: Running Example



(a) **(b)**

Figure 2: Landmark-trees for the running example

and as part of the value of the movie title (red occurrence in the first page). The limited set of available tags makes the former type of ambiguity more likely than the latter: HTML is an ambiguous language that allows distinct occurrences of the same tags and strings to play completely different roles in different regions of same page. While very often HTML tags are part of the template and texts are the actual content, the example shows that significant exceptions might arise.

The problem is further made more difficult by the fact that the characteristics of HTML code from templated sites are relatively local and might greatly change even from region to region of the same page. One region containing rigidly structured data records might come close to another one that contain much less regularly structured fragments of user-generated contents, or HTML snippets provided by external ads services, or (possibly nested) optional portions. As not every region of the page has to exhibit the same level of regularity in its structure, separating well structured regions from those with many different variants is a challenging task.

In this paper we show that the state-of-the-art in Web data extraction can be improved by giving specific attention to the HTML ambiguity problem. At the same time we assume rather simple structures for the organization of data, that is, we do consider nested tuples with optional attributes, but without nested lists [2, 14]. In practise, and thanks to the redundancy of large templated websites, this is seldom a severe limitation: websites publishing snippets of data organized into list of records, for example as search result pages, do repeat the same data organized in flat tuples in detail pages [29]. This is due to the fact that pages have to be consumed by humans, and their readability might be compromised by an overly complex structure.

Conversely, multi-valued attributes are simple list structures frequently found in Web pages, e.g., a list of author names separated by commas. Our approach can already extract the string spanning the whole list of values, and could be easily extended with a post-processing step to find the separator and split the values. However, dealing with lists is beyond the scope of the present paper, in which we focus on the minimal formalism necessary to present and solve the HTML ambiguity problem.

2 OVERVIEW

Web wrappers [4, 35] are software programs designed to extract data from the HTML source of Web pages. Here we give an idea of how we use formal grammars, a well established formalism in the foundations of computer science, to specify Web wrappers. We assume basic knowledge of the terms such as *terminal* (or *token*) and *nonterminal* in this context.

We use the term *Extraction Grammars* to refer to formal grammars where some nonterminals, which we call *extracting* nonterminals, play a specific role, to indicate that the strings of tokens deriving from such nonterminals are the data to be extracted (which vary for page to page) and not the elements of the template (which are repeated across pages).

Here we give some intuition on how they are used: Figure 1c shows an HTML template corresponding to pages such as those in Figures 1a and 1b with the extracting nonterminals E_{title}, E_{price} and E_{descr}, each named after the semantics of the extracted data shown in Table 1.

A major contribution of this paper is the family of *Landmark Grammars*, a new family of extraction grammars for modeling the HTML source code of pages from templated websites. Let us briefly illustrate them. Figure 2a shows a first example of landmark grammar in its compact representation, a ternary tree called *landmark-tree*. Each node is either a terminal symbol or an extracting nonterminal of the landmark grammar. Each edge has an orientation, used in the figure, based on the role of the child subtree, which can be either left or inner or right with respect to its parent. The tree need not be complete, as intermediate nodes have up to three sub-trees, but each of them has its distinct orientation. For example, `<B>` is the left child of the `<SPAN>` node; `Title` is the inner child of `<B>`, and `<DIV>` is the right child of `<B>`. There are a few leaf nodes marked with extracting nonterminals.

The core intuition behind landmark grammars is that the HTML source code of a collection of templated pages includes at least one token of the template that appears once and only once in each page. We call such a token a *landmark*, as we use it to split the page in at most three different *regions*, (i) the tokens before the landmark occurrence, (ii) those within it, and (iii) those after it. Landmarks correspond to pieces of the template, and the recursive split of pages allows us to handle HTML ambiguity, as repeated occurrences eventually end up in separated regions. Our observations and experiments over real websites, and a simple probabilistic analysis, which we summarize in Section 4.1 in a so called *Landmark Lemma*, all confirm that this intuition holds, and allows the generation of effective Web wrappers.

Let us comment on the various concepts by means of our example in Figure 1. In the two pages (as well as in many other similar pages obeying to the same HTML template) the token `Descr.` appears

exactly once and can play the role of landmark (the root of the landmark-tree in Figure 2b). On the contrary, the token `Price` cannot be chosen as a landmark of the whole page because it occurs twice in the page of Figure 1a: on the left of `Descr.` as part of the template as well as on its right as part of the content (as it is part of the title of the movie). However, it appears exactly once in the region before the only occurrence of the first landmark `Descr`. Recursively, when that landmark `Descr.` has been already posed, `Price` can be chosen as a landmark of its left subregions. This is represented in Figure 2b where `Price` is the left child of the root landmark `Descr`. An observation to be made is that the recursive construction of the tree ends for the regions in which no landmark is found: each region with no landmark gives rise to an extracting nonterminal.

Let us observe that the two alternative landmark-trees shown in Figure 2 for the pages in Figure 1 contain a different number of nodes: the tree in Figure 2b contains less nodes than that in Figure 2a and extracts a larger number of tokens. The two landmark grammars differ in the granularity of the data they extract. Indeed, the extraction in Table 2 shows that the nonterminals E_1, E_2 and E_3 in Figure 2b extract tokens that are part of the HTML template.

The example allows us to comment on an interesting formal problem addressed (and solved) in this paper: given a set of input pages obeying to a shared HTML template (for example those shown in Figure 1), find, among all possible extraction grammars (such as those shown in Figure 2), the one minimizing the length of the extracted strings.

Indeed, the fewer the tokens in the extracted strings, the greater is the number of tokens in the template. This has two consequences: more common portions of the template are recognised and the grain of the extracted data is finer, so improving the chances of giving them the appropriate semantics.

Finally, landmark grammars exhibit the interesting *local parsability* property, which allows the extraction process to gracefully degrade when mismatches between wrapper and pages occur. Parsing failures are confined as much as possible to allow the extraction of data from other regions of the page. This property has important practical consequences, as templated pages might come with many variations and exceptions that cannot be foreseen or observed when the corresponding wrapper is generated, for instance because the training sample used for the inference is too limited or biased.

The rest of this paper is organized as follows: Section 3 formalizes the Smallest Extraction Problem and presents landmark grammars, a family of extraction grammars supporting Web data extraction and exhibiting the local parsability property. Section 4 presents the problem of finding a landmark grammar matching with an input set of sample pages. Section 5 presents an optimal algorithm for solving the Smallest Extraction Problem. Section 6 introduces the dynamic tokenization technique to deal with HTML ambiguity. Section 7 presents the experimental evaluation of a prototype implementation of the proposed landmark-grammar inference algorithms. Related works are discussed in Section 8.

3 PROBLEM DEFINITION AND LANDMARK GRAMMARS

Following common practice [27], we define a context-free grammar G as a 4-tuple $\langle V_T, V_N, P, S\rangle$, where V_T is the terminal alphabet,

Table 1: Extraction using the landmark-tree of Figure 2a

Page	E_{title}	E_{price}	E_{descr}	Sum
a	`The Price of Everything`	`35.22`	`A US documentary`	8
b	`Star Wars`	`9.98`	`The saga directed by <B>G. Lucas</B>`	11

Table 2: Extraction using the landmark-tree of Figure 2b

Page	E_{title}	E_{price}	E_{descr}	E_1	E_2	E_3	Sum
a	`The Price of Everything`	`35.22`	`A US documentary`	`<B>`	`</B><DIV>`	`</DIV>`	12
b	`Star Wars`	`9.98`	`The saga directed by <B>G. Lucas</B>`	`<B>`	`</B><DIV>`	`</DIV>`	15

V_N is the nonterminal alphabet, P is the production set and S the axiom. A production (also called rule) is denoted by $A \rightarrow \alpha$, where $A \in V_N$ and $\alpha \in (V_T \cup V_N)^*$. In such a production, A (which is a nonterminal) is called the left-hand side (l.h.s.) and α (a sequence of terminal and nonterminal symbols) is the right-hand side (r.h.s.). The r.h.s. can be empty, in which case it is denoted by ε.

We use $A \Rightarrow \alpha$ to denote the transitive closure of the binary relation associated with the production set; $L(G)$ denotes the language associated with a grammar G, that is, $L(G) = \{s \in V_T^* | S \Rightarrow s\}$ is the set of strings of terminal symbols that can derived from the axiom S of the grammar.

As we said in the Overview section, we use grammars to describe wrappers extracting data from Web pages. A tokenizer performs a lexical analysis of the HTML source code, translating a page into a sequence of tokens (terminals in grammar terminology) from the alphabet V_T. Namely, the tokenizer creates one token for each word and discards portions of the input source which are neither tags or nor texts, such as HTML comments. As for tags, it produces two distinct tokens for a start tag and its closing counterpart.

Grammars for extractions. We use the term *Extraction Grammars* to refer to formal grammars coming with an attached semantics meant to specify wrappers for extracting data from Web pages. Some nonterminals of these grammars are specified to be *extracting nonterminals*: given a page, a wrapper extracts the strings of tokens deriving from such nonterminals.

Let us give a brief formalization of the notion. Given a nonterminal E in a grammar G and a page p, we use $E(p)$ to denote the sequence of tokens deriving from E when parsing p with grammar G. If $\mathcal{E}(G)$ is the set of the *extracting* nonterminals in G, then the semantics of G as an extraction grammar (to which we refer as an *extraction*), is the set of productions that derive the strings of tokens in the page from the extracting nonterminals $\mathcal{E}(G)$. In symbols, the extraction is the set $\{E(p) \mid E(p) \in {V_T}^+ \text{ and } E \in \mathcal{E}(G)\}$.

As we argued in the overview, in the realm of extraction grammars, an interesting goal is that of finding the one that allows for extraction at the finest grain. Let us now give a formalization of the problem, where the $|.|$ notation denotes, as usual, the number of tokens (that is, the length) of a string.

Definition (*Smallest Extraction Problem* — SEP)**.** Given a set of pages $\mathcal{P}$ and a family of extraction grammars $\mathcal{G}$, find the grammar $\overline{G} \in \mathcal{G}$ such that $L(\overline{G})$ includes all pages in $\mathcal{P}$ and $\overline{G}$ minimizes (the sum of) the sizes of the extracted strings:

$$\overline{G} = \arg\min_{G \in \mathcal{G}} \sum_{p \in \mathcal{P}} \sum_{E \in \mathcal{E}(G)} |E(p)|.$$

In this paper, we focus on a specific family of extraction grammars that exhibit the local parsability property [53], which allows the confinement of parsing failures.

3.1 Landmark Grammars

We introduce a family of extraction grammars called *Landmark Grammars*, discuss their properties and how they are used to extract data from pages. Then in Section 4 we will show how grammars of this family can be generated starting from a set of pages obeying to the same HTML template.

We make use of a notation to specify the productions composing a landmark grammar, which, at the same time, associates them with the paths of the corresponding landmark-tree: Given any node associated with a path labelled α, let $\alpha\triangleleft$, $\alpha\triangle$, and $\alpha\triangleright$ denote the path of its left, inner and right child, respectively. For example, in Figure 2a the path of landmark <B> is: $\triangleleft\triangleleft$.

A landmark grammar includes the axiom $S \rightarrow A_\varepsilon$, and specifies as extraction nonterminals all and only those occurring in the r.h.s. of the unit-rules, i.e., productions in the form $A_\alpha \rightarrow E_\alpha$. For each landmark node of the tree having path α, a number of productions are generated into the landmark grammar, as follows.

(a) if the landmark is a non-empty tag,[2] then it is associated with three productions, as follows:
$A_\alpha \rightarrow A_{\alpha\triangleleft} l_\alpha A_{\alpha\triangle} \bar{l}_\alpha A_{\alpha\triangleright} | E_\alpha | \varepsilon$
where l_α is the (opening) tag of the landmark and $\bar{l}_\alpha$ is the corresponding end tag;

(b) if the landmark is an empty tag (such as
) or a piece of text, then the productions have the form:
$A_\alpha \rightarrow A_{\alpha\triangleleft} l_\alpha A_{\alpha\triangleright} | E_\alpha | \varepsilon$
where l_α is again the landmark (with text form in this case);

(c) the productions $E_\alpha \rightarrow tE_\alpha | t$, where $t \in V_T$.

It is worth noting that each landmark is therefore associated with three productions, the first of which splits the page around a landmark occurrence whereas the other two are instrumental in extracting data, and handling parsing failures due to missing portions of the page, respectively.

Example 1. The landmark grammar corresponding to the tree in Figure 2a contains the following productions:

$S \rightarrow A_\varepsilon$
$A_\varepsilon \rightarrow A_\triangleleft$
$A_\triangleright | E_\varepsilon | \epsilon$
$A_\triangleleft \rightarrow A_{\triangleleft\triangleleft}$<SPAN>$A_{\triangleleft\triangle}$</SPAN>$A_{\triangleleft\triangleright} | E_\triangleleft | \epsilon$
$A_{\triangleleft\triangleleft} \rightarrow A_{\triangleleft\triangleleft\triangleleft}$<B>$A_{\triangleleft\triangleleft\triangle}$</B>$A_{\triangleleft\triangleleft\triangleright} | E_{\triangleleft\triangleleft} | \epsilon$
$A_{\triangleleft\triangleleft\triangle} \rightarrow A_{\triangleleft\triangleleft\triangle\triangleleft}$Title$A_{\triangleleft\triangleleft\triangle\triangleright} | E_{\triangleleft\triangleleft\triangle} | \epsilon$
$A_{\triangleleft\triangleleft\triangleright} \rightarrow A_{\triangleleft\triangleleft\triangleright\triangleleft}$<DIV>$A_{\triangleleft\triangleleft\triangleright\triangle}$</DIV>$A_{\triangleleft\triangleleft\triangleright\triangleright} | \ldots$
$A_{\triangleleft\triangleleft\triangleright\triangle} \rightarrow A_{\triangleleft\triangleleft\triangleright\triangle\triangleleft}$Price$A_{\triangleleft\triangleleft\triangleright\triangle\triangleright} | \ldots$
$A_{\triangleleft\triangle} \rightarrow A_{\triangleleft\triangle\triangleleft}$Description$A_{\triangleleft\triangle\triangleright} | E_{\triangleleft\triangle} | \epsilon$
$A_{\triangleleft\triangleright} \rightarrow A_{\triangleleft\triangleright\triangleleft}$<DIV>$A_{\triangleleft\triangleright\triangle}$</DIV>$A_{\triangleleft\triangleright\triangleright} | \ldots$
$A_{\triangleleft\triangleright\triangle} \rightarrow E_{\triangleleft\triangleright\triangle}$
$A_\triangleright \rightarrow A_{\triangleright\triangleleft}$<SPAN>$A_{\triangleright\triangle}$</SPAN>$A_{\triangleright\triangleright} | E_\triangleright | \epsilon$
$A_{\triangleright\triangle} \rightarrow E_{\triangleright\triangle}$
$A_{\triangleright\triangleright} \rightarrow E_{\triangleright\triangleright}$

For the sake of brevity, we do not show a few rules of the form $A_\alpha \rightarrow \varepsilon$ producing empty strings and rules for the generation and expansion of nonterminals E_α (those of the form $E_\alpha \rightarrow tE_\alpha | t$).

3.2 Local Parsability

We show that landmark grammars exhibit the *local parsability* property, which they directly inherit from the family of grammars in which this property has been originally introduced and developed: the *operator precedence grammars* [21, 52]. This property confines the effects of parsing failures to a local context. In our case, this applies to failures due to irregularities and variations in the page.

Local parsability is well exploited, for example, by the parsers underlying many IDEs. To provide the developers with real-time feedback even while working on large code bases, modern compilers are driven by the IDE to *incrementally* isolate and process only a limited portion of source code, as soon as the developer finishes typing it. For C-like programming languages, this can be reduced to parsing and recompiling only the smallest sequence of instructions contained by the pair of curly brackets surrounding the latest changes.

Mismatches between an automatically inferred grammar description of the HTML template and a page which is supposed to obey to that template are frequent and cannot be always avoided, due to several factors, some of which might be inherently part of the process used to collect the training pages: a sample might be not rich enough to let the inference algorithm observe all the fine-grained details of a complex template; or the presence of optional portions in the pages (e.g., ads), with a reserved and fixed room in the page layout, but without a predetermined structure; or even changes over time, i.e., new sections of the pages have been added, removed, or altered w.r.t. the version observed at inference time.

Considering all these challenges, we show that landmark grammars enjoy the parsability, which can confine the effects of these uncertainties with important practical consequences [52].

Proposition. Landmark languages are a strict subset of operator precedence languages.

Proof. For a grammar to be an operator precedence grammar, every production has to be in *operator-form*, i.e., the r.h.s. cannot contain two consecutive nonterminals, and cannot be empty. The productions of landmark grammars are not in the *operator-form* because they include productions with an empty r.h.s.. However, a simple bottom-up substitution allows us to convert any landmark grammar into a proper operator grammar. Namely, every production $A_\alpha \rightarrow \gamma A_{\alpha.x} \omega$ (with $\gamma, \omega \in (V_T \cup V_N)^*$) in which $A_{\alpha.x}$ occurs in the r.h.s. is replaced by two new productions of the form $A_\alpha \rightarrow \gamma\omega$ and $A_\alpha \rightarrow \gamma A'_{\alpha.x} \omega$. $A'_{\alpha.x}$ r.h.s. is obtained from A_α's by replacing every A nonterminal with the corresponding A'. This procedure removes productions with empty r.h.s. and it does not lead to any new production having two consecutive nonterminals in the r.h.s.

The containment is strict because the operator grammars can be recursive whereas the landmark grammars productions are not, except for the extracting nonterminals, defined as $E_\alpha \rightarrow tE_\alpha | t$. □

We now show that the local parsability can enable the generation wrappers that *locally* confine the effects of parsing failures, e.g., due to irregularities of any of the kinds mentioned above. To the best of our knowledge it has never been exploited for this purpose.

Parsing with Error Handling. Let us assume now that a landmark grammar is available. In Section 4 we illustrate an algorithm for inferring such a grammar starting from a set of pages sharing a common HTML template.

The parsing of a page with a landmark grammar can be executed by means of a non-directional parser [27, 52]. The parsing algorithm

[2] For the sake of simplicity we ignore all tags' attributes and their values by considering all tag tokens made up only by their element name. We defer further discussions on the inner structure of tag tokens to Section 6.

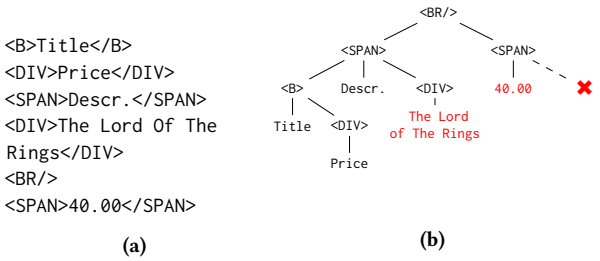


Figure 3: Extraction in the running example

performs a top-down visit of the landmark-tree and a contextual decomposition of the input page into *regions*, where a region is a string of contiguous tokens within the page. During such visit, it considers the productions having in the l.h.s. the landmark in current tree node.

Given a node with path α of the landmark-tree, let the corresponding nonterminal A_α be associated with a region r_α (initially A_ϵ is associated with r_ϵ spanning the whole input page p).
The parsing is executed as follows:

(1) apply the production $A_\alpha \rightarrow A_{\alpha\triangleleft} l_\alpha A_{\alpha\triangle} \bar{l}_\alpha A_{\alpha\triangleright}$ if l_α is a tag, (or $A_\alpha \rightarrow A_{\alpha\triangleleft} l_\alpha A_{\alpha\triangleright}$ if it is a text) by searching in r_α for an occurrence of the landmark l_α;
   (a) halt if either none or more than one occurrence are found: the string of terminal symbols in r_α are extracted by parsing nonterminal E_α by applying the production $A_\alpha \rightarrow E_\alpha$;
   (b) otherwise (exactly one occurrence l has been found) split r_α into three sub-regions $r^l_{\alpha\triangleleft}$, $r^l_{\alpha\triangle}$ and $r^l_{\alpha\triangleright}$ w.r.t. l; recursively repeat the procedure on each of the sub-regions of r_α and the corresponding sub-regions by parsing $A_{\alpha\triangleleft}$, $A_{\alpha\triangle}$, and $A_{\alpha\triangleright}$ over $r^l_{\alpha\triangleleft}$, $r^l_{\alpha\triangle}$ and $r^l_{\alpha\triangleright}$, respectively.

Example 2. Figure 3a shows a page obeying to the HTML template shown in Figure 1c but missing the movie *Description*. This does not prevent the landmark grammar shown in Figure 2a from extracting other data from the nodes marked with E_{title} and E_{price} as shown in Figure 3b ("X" stands for the missing node E_{descr}).

3.3 Split Operation

Given a region r, and one token t occurring exactly once within it, we can define a three-way *split* operation of that region into three contiguous sub-regions based on that unique t occurrence: the region at the left ($r^t_\triangleleft$), within ($r^t_\triangle$, if t is tag), and at the right ($r^t_\triangleright$) of t. The split operation is well-defined if and only if exactly one occurrence of the token t can be located in r. By extension, we consider it well-defined also if no occurrence is found: the semantics is that it just returns the region on which it is applied.

A split operation can be associated with a production of a landmark grammar: let A be a nonterminal in the l.h.s. of a production from which the string of tokens in a region r derives. It turns out that the regions deriving from $A_\triangleleft$, $A_\triangle$ and $A_\triangleright$ can be seen as resulting from a split operation pivoting on a landmark l occurring exactly once in the region r.

Algorithm 1 FINDTEMPL: Creating the landmark-tree

```
1: function FINDTEMPL(R)                                   ▷ Input: a set of regions.
2:   l ← choose a landmark occurring at most once in every r ∈ R
3:   return l(FINDTEMPL(R^l_◁); FINDTEMPL(R^l_△); FINDTEMPL(R^l_▷))
4: end function                                            ▷ Output: a landmark-tree.
```

Example 3. If we consider the region r spanning the whole page in Figure 1a, then the following regions are obtained by splitting on the token
:
$r^{\texttt{
}}_\triangleleft$ = <B>Title</B>...<DIV>The Price of Everything</DIV> ;
$r^{\texttt{
}}_\triangle = \epsilon$;
$r^{\texttt{
}}_\triangleright$ = <SPAN>35.22</SPAN> A US documentary .

Parsing a landmark grammar reduces to recursively applying split operations that consume and produce page regions. This observation motivates the following complexity analysis.

Proposition (Split Complexity)**.** The worst-case time complexity of a split operation over a set of regions containing n total tokens can be implemented in $O(n \log n)$.

Proof. We assume to have built a *token-table* structure over all the tokens in the input regions. The token-table is a map associating each token with the ordered list of all its occurrences over the input regions, and can be built in $O(n)$. These lists are ordered by the position of the token occurrences in the page. In order to perform a split operation on a certain pivoting token, we need to compute the resulting sub-regions by splitting the token-table in several smaller tables each associated with the resulting sub-region, so as to support next split operations. The split positions of the pivoting token can located in the ordered lists of occurrences associated with any other tokens (since those lists need to be split, as well) in $O(\log n)$. The number of lists to split in the worst case is $O(n)$. □

4 WRAPPER INFERENCE

In the previous section we have described how we can parse a given landmark grammar to extract data from a page. In this section we show how such a grammar can be synthesized.

Given a set of input pages, Algorithm 1 (FINDTEMPL) aims at generating the landmark-tree representation of a landmark grammar that describes the HTML template underlying every input page. The produced grammar can be used as the specification of a Web wrapper aiming at fine-grained data extraction from other pages with the same template.

Algorithm FINDTEMPL builds a landmark-tree representation of the output grammar starting from its root in a top-down fashion. As the first invocation triggers the overall inference process, it receives as input a set of regions, each spanning one full page. The root of the output landmark tree is then chosen by analyzing the content of every input region (line 2) looking for tokens occurring at most once in every input page, if any.

The input parameter (R) of the algorithm is one set of regions, each from one distinct page of the input set of sample pages. Each recursive invocation of the algorithm reduces the problem by splitting every input region r on the landmark occurrences l it contains, if any: the sub-regions $r^l_\triangleleft, r^l_\triangle, r^l_\triangleright$ are computed as detailed in Section 3.3. Left, inner, and right sub-regions are then grouped so as

to create three new sets of regions, namely $\langle R^l_\triangleleft, R^l_\triangle, R^l_\triangleright \rangle$, on which the algorithm is recursively invoked (line 3).

The parenthesis notation (r.h.s. of line 3) stands for an operator $l(\cdot,\cdot,\cdot)$ constructing the ternary landmark-tree rooted in the chosen landmark l, and having three children, which are the roots of the sub-trees resulting from recursive invocations.

Example 4. Two of the possible landmark-trees inferred for our running example are depicted in Figure 2. They can be obtained from the input pages shown in Figures 1a, 1b, and 3a.

Solving the overall inference problem is equivalent to recursively solving several instances of the *Landmark Finding Problem*.

Definition (The Landmark Finding Problem — LFP)**.** Given a set of input regions (one per page) assumed to obey to the same HTML template, choose which landmark candidate should become the next landmark of a grammar describing every input region.

Algorithm findTempl uses a nondeterministic operator **choose** to solve the inference problem. It considers as a legit landmark candidate any token occurring at most once per page, that is, any token unambiguously supporting a split operation over *every* input region. Therefore, it ends up considering all the possible ternary landmark-trees modelling the set of input pages. In the worst-case, the input contains n different tokens, and the number of possible landmark-tree can become as large as the number of possible ternary trees with n nodes, $C^{(3)}_n = \frac{1}{2n}\binom{3n}{n}$, i.e., the generalized Fuss-Catalan number [3].

Proposition (findTempl complexity)**.** The worst-case complexity of enumerating all the landmark-trees for an input made of n total tokens is $O(C^{(3)}_n \cdot n^2 \log n)$.

Proof. It follows from Split Complexity, and from the cost of building one of the possible $C^{(3)}_n$ landmark-trees of n nodes by means of n split operations. □

A deterministic solution is presented in Section 5, where we also explain how to build, among all produced landmark-trees, the one that solves the SEP if we pose additional restrictions on the occurrences of each landmark. Next we show, by developing a simple probabilistic model of modern templated HTML pages, that the intuitive and effective heuristic of choosing the candidate landmarks amongst all tokens occurring at most once per input page is also well principled. Indeed, that is the property characterizing the tokens that most likely are part of the template according to the developed model.

4.1 Landmark Lemma

Pages of modern sites are generated by scripts that query one or several underlying databases. We assume that no information is available about those scripts beyond an observed input set of regions R (possibly, whole pages).

A token occurring in the input set of regions can be classified in two mutually exclusive categories: either it is a *template* token or a *value* token, i.e., either it is produced by the script as part of the HTML template, or it comes from the contents stored in the underlying databases. Nevertheless, without a direct access to those generating scripts, such classification can only be approximated by developing a probabilistic model based on the evidence collected while observing the input set of regions R.

We introduce a discrete probability distribution function $p(\cdot)$ over tokens in the input regions R such that $p(t,R)$ is the probability that all occurrences of t in R are template tokens, and so $1-p(t,R)$ is the probability that at least one of them is a value token.

We develop a workable probabilistic model based on two assumptions. The first is the following.

Assumption (Independence of Occurrences — IOA)**.** Given a token t, and a set of regions R, $p(t,R)$ depends only on the nature of the token t (i.e., whether it is a tag or a text) and on the number of its occurrences in every input region $r \in R$.

The IOA can be rewritten as $p(t,R) = p(t,o(t))$ where $o(\cdot)$, called *occurrences vector* [2], is a function over tokens in R that associates a token with the number of its occurrences in every input region $r \in R$.

This assumption is not completely valid on real Web pages as tags are often organized according to an HTML language specification that "ties" together certain tags, say, <TR> and <TD> for creating table rows. We end up neglecting, in our model, how those occurrences are positioned w.r.t. occurrences of the same token and of other tokens in the same regions.

The second assumption is that every tag is part of the template with the same *template probability* $\bar{l}$, and every text with probability $\bar{v} = 1-\bar{l}$ or vice versa, i.e., a text is not part of the template with probability $\bar{l}$ and is template with probability $1-\bar{l}$. This allows us to simplify our probabilistic model into a single-parameter one. In practice, we can expect the parameter $\bar{l}$ to be small: usually most of tags are part of the template and most of the texts are values.

We are now ready to state and prove a major property that justifies the role of landmark grammars as an effective formalism for describing HTML templates.

Lemma (Landmark Lemma)**.** Given a set of input regions R, if *template probabilities* are either very small ($\lim_{\bar{l}\to 0+}$, i.e., texts) or very large ($\lim_{\bar{l}\to 1-}$, i.e., tags), then the occurrences vector that maximizes the probability of a token t being a landmark is the *binary vector*, for which the token does not occur more than once in any input region.

Proof. The probability of a token t being part of the template can be computed by the mutually exclusive events of it being found as an occurrence of a template token i times (with probability $\bar{l}$), and $k-i$ times as an occurrence of a value token (with probability $\bar{v} = 1-\bar{l}$), *i.e.*:

$$p(t) = \sum_{i=1}^{k} \bar{l}^i\, \bar{v}^{k-i} = \frac{\bar{l}\left(\bar{l}^k - \bar{v}^k\right)}{\bar{l}-\bar{v}}\,. \tag{1}$$

Our goal here is to find the maximum value of the probability, and hence we compute the partial derivative of $p(\cdot)$ w.r.t. k:

$$\frac{\partial p}{\partial k} = -\frac{\bar{l}\left((1-\bar{l})^k \log(1-\bar{l}) - \bar{l}^k \log(\bar{l})\right)}{2\bar{l}-1}\,, \tag{2}$$

where we need to assume $\bar{l} \neq 0, \bar{l} \neq 1, \bar{l} \neq \frac{1}{2}$. By solving for k:

$$\frac{\partial p}{\partial k} = 0 \longrightarrow k = \frac{\log\left(\frac{\bar{l}\log(\bar{l})}{\log(1-\bar{l})}\right) - \log(\bar{l})}{\log(1-\bar{l}) - \log(\bar{l})}\,, \tag{3}$$

and, by taking the limit for $\bar{l} \to 0+$ (for tags) and for $\bar{l} \to 1-$ (for texts), we obtain the following result:

$$\lim_{\bar{l}\to 0+} k = \lim_{\bar{l}\to 0+} \frac{\log\left(\frac{\bar{l}\log(\bar{l})}{\log(1-\bar{l})}\right) - \log(\bar{l})}{\log(1-\bar{l}) - \log(\bar{l})} = \lim_{\bar{l}\to 1-} k = 1. \quad (4)$$

We now prove by induction that it holds for n pages. The basic step follows from Equation 4 above. As for the inductive step, let us now assume that the property holds for $n-1$ pages by inductive hypothesis for a candidate token t. Given a new page, either that candidate does not occur at all and cannot be observed (i.e., $k = 0$), or, if it occurs, it has to be present at most once ($k \leq 1$) so that Equation 4 holds, as well. □

5 SEP AS A SEARCH PROBLEM

The family of landmark grammars allows the SEP to be conveniently modeled as an informed search problem [54] in a space whose states are each associated with a landmark-tree. Every edge of the space is associated with a possible split on the landmark-tree of the state it departs from, and the split creates the landmark-tree associated with the state the edge leads to.

SEP is thus reduced to the problem of finding the state s^* with the smallest cost, the latter being defined as the total size (in tokens) of the strings extracted by its underlying landmark-tree from the input pages: $g(s) = \sum_{r\in R(s)} |r|$, where $R(s)$ is the set of all regions extracted by the landmark grammar associated with the state s. The initial state of this search space is the empty landmark-tree whose cost is equal to n, the total number of tokens in the input pages. The goal states are the landmark-trees on which additional splits cannot be operated for the absence of candidate landmarks. SEP is thus reduced to the problem of finding the goal state s^* with the most fine-grained data extraction, i.e., the smallest cost $g(s^*)$.

Example 5. Figure 4 shows an excerpt of the search space for the running example. The root is an empty landmark-tree. The two highlighted landmark-trees are those in Figures 2a and 2b. Several paths might lead to the same state as shown by the gray colored state in Figure 4: it is possible that a landmark-tree is obtained by applying the same set of split operations in different sequences.

The number of possible states can be overwhelming, even for rather small input pages, if each contain different tokens. However, for real pages, and especially templated ones, many tokens are repeated in every page. The efficiency of the search algorithm can be measured with the number of number of visited states, i.e., the number of splits performed to reach a goal state.

We introduce k-SEP, a constrained variant of the general SEP specifically designed for the family of landmark grammars. It reduces the number of states of the search space by discarding tokens not occurring frequently enough to be considered as plausible landmarks. It is based on the following additional assumption.

Assumption (k-SEP)**.** Given a landmark grammar and a set of input regions that it parses, every landmark must occur exactly once in at least k input regions.

It is worth mentioning that FINDTEMPL actually deals with 1-SEP, as it implicitly requires every landmark to occur at least once to be observed and considered as one of the possible candidate landmarks.

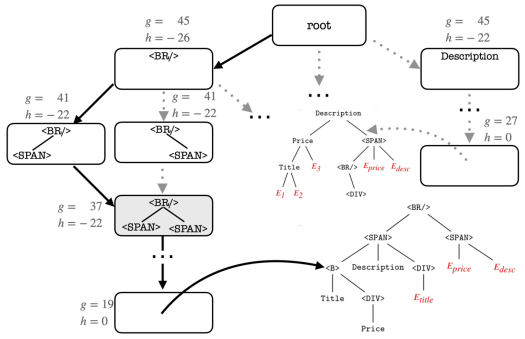


Figure 4: Excerpt of the search space for the example

5.1 A*-Solution for k-SEP

The search space of a k-SEP instance features two important characteristics: first, a goal state is any state associated with a landmark-tree on which no further split is possible for the absence of candidate landmarks satisfying the k-SEP assumption; second, any split decreases the overall cost of at least k units, as that is the minimum number of token occurrences it removes from the extracted strings.

A* is a best-first search algorithm [54] that maintains a frontier over all and only the visited states that could still potentially be in the path to the optimal goal state. In every iteration, and under rather general assumptions on the adopted cost function, it selects which paths departing from that frontier need to be further expanded towards the goal to not miss the optimal state.

A* bases its decisions on a cost function $f(s) = g(s) + h(s)$, that aims at optimistically estimating the cost from the initial state to the optimal goal state by crossing the current state s: $g(s)$ is the actual cost of the path to reach the current state s, and the *heuristic* function $h(s)$ estimates the cost required to expand the path from s all the way down to the optimal goal state.

We define $h(\cdot)$ as the function estimating the number of tokens yet to be modelled by a landmark-tree, as $h(s) = i(s) - g(s)$, where $i(s) = o_{<k}(s) + o_u(s)$. $o_{<k}(s)$ counts the number of tokens occurring in less than k regions; $o_u(s)$ is the number of tag occurrences which are left "unbalanced" by a previous split: they are opened (resp. closed) in a region but closed (resp. opened) in a different one. We name $i(s)$ *incompressible cost* after the observation that it counts the tokens that cannot be modelled (from there on) in a k-SEP search space by a landmark. With this definition of the heuristic function, it is the incompressible cost function $i(\cdot)$ that ends up driving A*'s visit of the search space. Indeed, $f(s) = g(s) + h(s) = i(s)$.

Example 6. Figure 4 shows $g(\cdot)$ and $h(\cdot)$ values over a few states of the 2-SEP search space for the running example. The A*-algorithm follows the path that includes the nodes on the left side of the figure and reaches the optimal landmark-tree, which is the state at the bottom left of the figure.

It is well known that A* algorithm effectiveness strongly depends on some crucial properties of the adopted heuristics function $h(\cdot)$.

As for its *completeness*, i.e., the capability of always finding the optimal solution, A* requires $h(\cdot)$ to be *admissible*. An heuristic $h(\cdot)$ is said admissible if it never overestimates the cost of reaching the

goal, i.e., $h^*(s) \leq h(s) \leq 0$ for every state, where $h^*(s)$ is the actual cost of the optimal path from the current state to the goal.

The A* algorithm is said *optimally efficient*, i.e., with the guarantee that there cannot exist other algorithms that always visit fewer states than A*, if $h(\cdot)$ is *consistent* (or *monotone*). An heuristic $h(\cdot)$ is said consistent if its value never decreases along any path from the initial state towards a goal state.

Lemma (Properties of the Heuristic Function)**.** The heuristic function $h(s) = i(s) - g(s)$ is consistent and admissible for $k \geq 2$.

Proof. We only prove the consistency as it also entails the admissibility. Given a state s, let z be any of its successor, and let s^* be the goal state; for a heuristic to be consistent, the following conditions must hold [49]:

(1) $h(s^*) = 0$;
(2) $h(z) \geq c(s, z) + h(s)$, with $c(s, z)$ being the actual cost of reaching a descendant z from s.

The first condition is trivial as the algorithm itself halts when only incompressible tokens are left in the extracted regions.

As for the second condition, when going from a state s to one of its children, say z, the cost of reaching z from s is:

$$c(s, z) = g(s) - g(z) = \sum_{r \in R(s)} |r| - \sum_{r \in R(z)} |r|,$$

and therefore we can rewrite that condition as:

$$i(z) - \sum_{r \in R(z)} |r| \geq \sum_{r \in R(s)} |r| - \sum_{r \in R(z)} |r| + i(s) - \sum_{r \in R(s)} |r|,$$

that is: $i(z) \geq i(s)$. The latter trivially holds by definition of the incompressible cost function $i(s) = o_{<k}(s) + o_u(s)$. A token counted by $o_{<k}(\cdot)$ occurs exactly once in less than k regions cannot occur more frequently by means of some additional splits. Similarly for $o_u(\cdot)$: unbalanced tags of the regions extracted in the current state cannot later be rejoined by a split. □

5.2 Greedy Search

Although the worst-case time complexity for inducing an optimal landmark tree by means of the A* algorithm is still exponential, real pages can hardly trigger such exponential behaviour in practice. Indeed, backtracking can only occur in the presence of tokens in the input regions with a recurrent HTML ambiguity, when the behavior of template tokens is not distinguishable from that of value tokens by just counting the occurrences. Even if backtracking happens, it is most likely going to be "localized", i.e., confined in smallish regions, at tolerable costs.

We now introduce Greedy, a variant of the search algorithm that removes any trace of backtracking by always expanding the state with a minimal value of the cost function $f(\cdot)$: it can be used to benchmark the contributions of A* both in term of additional computational costs, and in term of the improvements over the quality of induced landmark-trees. It can also be used to study the characteristic of the regions that trigger the backtracking when running A* on real pages.

Greedy, as well as A*, tends to prefer landmarks with the largest number of occurrences amongst those occurring exactly once in at least k pages. However, whereas A* backtracks to already visited states for finding the best choice, Greedy's single choice might significantly affect the quality of the final outcome.

We also aim at measuring the importance of the tie-breaking policy by considering two Greedy variants: Greedy-rnd makes a random choice, whereas Greedy-inf makes an informed decision by adopting additional priority criteria, as follows: texts are better than tags; longer texts are preferred over shorter ones; tags higher in the DOM hierarchical representation of input pages are preferred over those deep in the tree.

These criteria for prioritizing the landmark aim at preferring the candidates that are most likely part of the template. They respond to the intuition that textual candidate landmarks, especially longer ones, are more likely to be part of the template than a leaf tag.

Independently of the adopted variant, Greedy certainly exhibits a better worst-case time-complexity than A*.

Proposition 7 (Greedy-findTempl complexity)**.** Greedy worst-case time-complexity for inducing a landmark-tree is $O(n^2 \log n)$.

Proof. It follows from the $O(n)$ nodes in a landmark tree, from the complexity $O(n \log n)$ of split operations, from the $O(\log n)$ cost of inserting new landmarks into a priority queue that maintains the candidates' ranking, and from the fact that every split can originate at most 3 new candidates to insert in the priority queue. □

6 HTML DYNAMIC TOKENIZATION

So far we neglected the *attributes* of tag tokens for the sake of simplicity. We now remove this limitation, and leverage the richness of modern pages arising from the rather complex inner structure of an HTML tag: each has an element name along with an optional list of attributes, each with a name and an optional value.

The source code of a templated page often contains many tags with the same element name but different roles in the template. The roles can be distinguished by also leveraging the differences in the attributes of a tag. For example, if `<DIV class="movie-title">` and `<DIV class="movie-price">` are modeled by the same landmark `<DIV>` that ignores their HTML attributes, we would end up confusing tokens that clearly play two distinct roles in a templated page: the landmarks need to be sensitive to the value of the attributes, say `class`, to distinguish the occurrences' roles.

Unfortunately, deciding which attribute names and values should be considered for every candidate landmark is a challenging problem. The presence of HTML attributes turns out to be occasionally misleading: consider for example the quite common practice of highlighting even and odd rows of a long HTML table with different colors to facilitate reading. This detail has nothing to do with the semantics of the contents: it is preferable a landmark `<TR>` that neglects the presence of the `bgcolor` attribute and so matches both `<TR>` and `<TR bgcolor="#808080">`.

Let `<T>` denote a landmark matching all tag tokens having as element name `T`, independently of the presence of attributes. We introduce the notation `<T a=* b=*>` for a landmark sensitive to the presence of attributes and matching the two occurrences `<T a="x" b="y">` and `<T a="z" b="z">`, and the notation `<T a="x" b=*>` for a landmark that is also sensitive to value of the `a` attribute: it matches with `<T a="x" b="y">` but not with `<T a="z" b="z">`.

A straightforward extension of our wrapper inference algorithm can deal with HTML attributes by reconsidering the SEP search

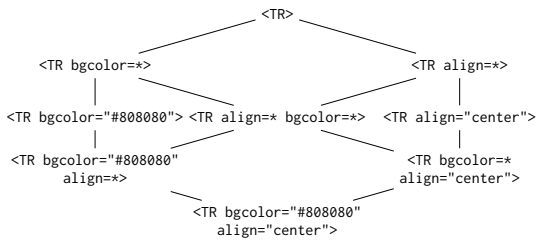


Figure 5: Lattice of tag landmark candidates

space modeling discussed in Section 5: the search space built without considering attributes is now expanded with states to take into account the new landmark candidates.[3]

This technique is named *dynamic tokenization*: at inference time, it deals with *lexical* aspects that are traditionally faced during the preliminary tokenization of the input pages, well before the inference even started. Namely, it decides which HTML attributes of HTML tags should somehow "emerge" into tokens, to better solve the k-SEP. In a sense, the induction algorithm is inferring not only a landmark grammar description of the input pages, but also, and contextually, the most appropriate lexical description of its tokens to let that inference succeed.

All the candidate landmarks matching with a tag token of a certain element name can be ordered into a lattice structure $(C, \preceq)$ as shown by the example lattice in Figure 5: it has been created with the candidate landmarks that model all `TR` occurrences, such as `<TR bgcolor="#808080" align="center">`.

Let C be the set of all possible candidates with the same element name and let $\preceq$ denote a partial order relation between candidates. The least upper bound of the lattice is always the landmark matching with all token occurrences having the same element name, independently of the presence of HTML attributes.

Given two landmarks T_1 and T_2 from that lattice: $T_1 \preceq T_2$ iff for every token occurrence t, t matches $T_1 \implies t$ matches T_2, i.e., landmarks that are greater according to the partial order $\preceq$ have the same or a larger number of occurrences in every page.

During the inference, in order to decide whether a token can play the role of a landmark or not, the tags occurring in the input regions are grouped by their element name, and each is associated with a lattice hosting all its occurrences. The candidates in the lattice having exactly one occurrence in at least k pages contribute with a state in the k-SEP search space.

We assume that only a finite and already known set of attribute names and values can contribute to the lattice:[4] A* visits more states when the dynamic tokenization is enabled because the new candidates increase the branching factor of the search space. However, both Greedy and A* worst-case time complexity do not increase.

[3]Conversely, extending the landmark grammars would turn out cumbersome as formal grammars assume an up-front static lexical analysis of the input string.

[4]Our prototype can be configured by specifying which attributes should be used to build the lattice (in our experiments: `id,class`), optionally with the value (`class`), or that should just be ignored (`bgcolor`).

7 EXPERIMENTAL EVALUATION

We developed a Java prototype implementation of findTempl for automatic wrapper inference based on landmark grammars. It implements the optimal algorithm A*, the two variants of the Greedy algorithm, a baseline Random that will be presented later, and the HTML dynamic tokenization technique.[5] Our experimental evaluation [6] is based on two benchmarks: (i) the Structured Web Data Extraction (SWDE) dataset [30]; (ii) the Alaska benchmark [10].

SWDE is a rather consolidated dataset progressively adopted as a reference by several Web data extraction systems [6, 30, 38, 42, 50] and therefore suitable for comparing existing approaches. It was sourced from 80 websites over 8 domains for a total of $124k$ pages.

Alaska is a more recent benchmark, which was conceived for Web data integration tasks, but can be used for evaluating Web data extraction systems as well. It includes a large manually curated ground-truth with 587 attributes from 20 websites.

Each of the benchmarks comes with a ground-truth made of correct attribute values from the pages of the dataset. To evaluate the quality of the generated wrappers, we compute the *precision* and *recall* over all attributes in the ground-truth. Given a collection of pages $\mathcal{P}$ and an attribute a, let $a(\mathcal{P})$ be the set of attribute values extracted from the pages by the wrapper for that attribute, and let $a_g(\mathcal{P})$ be the corresponding correct values for the same attribute from the ground-truth: precision is defined as $P = \frac{|a_g(\mathcal{P}) \cap a(\mathcal{P})|}{|a(\mathcal{P})|}$, and recall as $R = \frac{|a_g(\mathcal{P}) \cap a(\mathcal{P})|}{|a_g(\mathcal{P})|}$. Then, F-measure is computed as usual: $F = 2 \times \frac{P \times R}{P+R}$. In order to associate each ground-truth attribute with one of the output attributes, we always select the output attribute having the best F-measure value. As a measure of the cost, we count the number of split operations. For an approximate measure of the computation times, our sub-optimized prototype on a commodity machine[7] takes 4.36 seconds for $1k$ serial splits (by averaging over all the experiments we conducted). Over the SWDE dataset, A* performs an average of ~$14k$ splits per site (with a standard deviation σ =~$17k$); for the Alaska Benchmark it performs an average of ~$600k$ splits per site (σ =~$38k$). It is worth noticing that the presented inference algorithms can be easily decomposed to perform several splits in parallel.

7.1 SWDE Benchmark

SWDE has been used for the evaluation of several Web data extraction systems proposed in the literature. Moreover, it has been used for the evaluation of WADaR [41], which is a tool for joint wrapper and data repairing, itself based on the exploitation of several pre-existing proposals such as DEPTA [60], DIADEM [22], ViNTs [61], RoadRunner [14], and WEIR [6]. Indeed, WADaR fixes the data extracted by automatically generated wrappers and improves their quality (the F-measure grows by $15\% - 60\%$ [42]).

The main idea behind WADaR's repairs is to exploit the hints coming from a bunch of domain-specific annotators (*oracles*, in WADaR's terminology), which it uses to detect and fix anomalies

[5]We consider the attributes of all tags without their values, except for the attributes `class` and `id`, for which we also consider the values. `bgcolor` is ignored.

[6]The dataset, the results and the logs of the experiments are available at this address: https://thesmallestextractionproblem.github.io.

[7]Equipped with a 2.50GHz Intel Core i7-6500U processor and 8Gb of RAM

in the extracted tuple of values. For instance, it can check whether the extracted value, or a substring of it, belongs to the domain of an attribute. If not, WADaR will look for a repair transformation to jointly fix all the tuples of values extracted by the wrapper. Because of the very nature of WADaR, which improves the quality of the results of other systems, a comparison with WADaR transitively provides a comparison with all the systems it is based upon.

Two other recent systems have been evaluated over SWDE: HYB [50], a supervised wrapper generator, and OpenCeres [38], an *Open Information* system capable of automatically discovering new attributes. The evaluation of OpenCeres has motivated the definition of an *expanded* version of the benchmark, which includes additional ground-truth for 272 additional attributes from 21 sites of 3 SWDE domains (Movie, NBA, and University).

Table 3 shows precision and recall averaged over all the attributes in the SWDE ground-truth (or the expanded version, when available). We report the results for A* (with the dynamic tokenization enabled), WADaR, HYB, and OpenCeres.

A* has been evaluated by inferring a wrapper from a random sample of 20 pages for every site (0.01-0.1% of those available in the dataset). A* achieves 93.88% in precision and 93.25% in recall, improving WADaR's results by 12.00%, and 20.13%, respectively. It is worth noticing that WADaR requires external knowledge for its repairs, in the form of domain-specific annotators, while our entire process is fully automatic.

Table 3 also shows that A* outperforms HYB, a semi-supervised system to synthesize a wrapper from very few examples: the user is required to interact with the system to provide positive and negative examples that will drive the wrapper generation starting from a few initial annotations (just 2 on the SWDE dataset).

We also report the results of the evaluation over the *expanded* SWDE against OpenCeres, a system that requires a seed KB made of pairs of attribute value/label to run the extraction. The expanded dataset contains more challenging optional attributes w.r.t. the original dataset (with 12% loss both in precision and recall) but A* can still outperform OpenCeres. However, it is worth observing that OpenCeres has a different and more challenging goal, as it aims at finding both labels and values of attributes.

Finally, we also compare A* against Random and Greedy, the other sub-optimal algorithms visiting the same search space, as a baseline: as Random selects landmarks without a clue, it helps to assess how challenging a dataset is; Greedy is used as a reference to analyze the role of A*'s backtracking for its optimality.

7.1.1 Random. The results in Table 3 are obtained by averaging 100 executions over all the sites of each domain. Interestingly, and despite the totally clueless strategy, Random obtains respectable precision results (always above 73%) on SWDE. Conversely, the recall is rather disappointing (always below 48%), revealing that minimizing the number of false negatives is the real challenge for this benchmark. The results on the expanded dataset are similar.

7.1.2 Greedy. The greedy version of our inference algorithm visits the search space by always preferring the landmark that minimize the length of the extracted strings after the split. Table 3 shows the results obtained by Greedy-rnd and Greedy-inf, the two variants differing only for the policy adopted for breaking the ties. The former makes a random choice, while the latter heuristically ranks the candidates, as described in Section 5.2.

Table 3: Experiments on the SWDE dataset

	SWDE			Exp. SWDE		
	P	R	F	P	R	F
WADaR	81.88	73.12	77.25			
HYB	86±3	87±3	86±3			
OpenCeres				72	48.33	57.84
Greedy-rnd	91.25	90.63	92.27	78.33	78.67	78.50
Greedy-inf	92.38	90.88	91.62	80.33	79.67	80.00
A*	93.88	93.25	95.56	81.67	81.67	81.67
Random	77±2	34±3	48±1	64±5	12±2	20±1

The obtained results are close to those of the optimal algorithm A*, achieving a precision of 92.38% (resp. 91.25%) and a recall of 90.88% (resp. 90.63%) with Greedy-inf (resp. Greedy-rnd), i.e., a loss of just 2.06% in precision and 2.5% in recall. On average it saves 9.8% splits (with a standard deviation of 20.87%). Also, the differences in the results of the two variants show that only a rather small portion of the loss can be recovered by using smarter policies to break ties: most of the loss is due to the lack of backtracking.

As for the results on the expanded dataset, it is worth noticing that the differences between A*'s and Greedy's results are similar to those computed on the original SWDE dataset.

7.2 Alaska Benchmark

The Alaska benchmark is an end-to-end benchmark for Big Data Integration tasks. It contains more than 29*k* pages from 20 websites. The dataset was not specifically designed as a data extraction benchmark, and despite the SWDE dataset contains more pages and sites than Alaska's, the variety of attributes in the latter dataset makes it more challenging.

Indeed, let us classify attributes as either *head* or *tail* depending on whether they are present or not in the majority of the sites provided by a benchmark for a certain domain. The SWDE benchmark is by construction built in such a way that it includes only 3-5 head attributes per domain. Conversely, the majority of the attributes in the Alaska Benchmark's ground-truth are not head attributes.

Head attributes are usually located in a fixed, non optional portion of the template, while tail attributes are more challenging and optional attributes generally located in regions of the page with a looser HTML structure.

7.2.1 Alaska Benchmark Ground-Truth. The Alaska benchmark includes a dataset of Web pages about e-commerce products with a manually curated ground-truth made of linkages (pair of pages referring to the same real-world entity) and schema matches (pair of attributes from distinct sites with matching semantics). Unfortunately, it does not include a ground-truth specifically designed for Web data extraction, as the benchmark aims at evaluating the integration of the extracted data rather than their extraction from the HTML source of the pages. Indeed, the dataset is available as JSON files created by using an ad-hoc extractor specialized for HTML tables containing products specification.

Given the size of the benchmark, manually curating the quality of those extracted data is a daunting task. We derived a Web data extraction ground-truth by looking for the occurrences of attribute values in the HTML source code of the pages. We selected

only the attributes that according to the Alaska benchmark ground-truth are offered by several pages across distinct sites. Since the results of the integration is manually curated, and the data are redundantly offered by multiple sources, we build our Web data extraction ground-truth based on the assumption that these correctly integrated redundant data are correctly extracted, as well. We manually verified and confirmed the correctness of all the values for a random sample of 50 values. We also manually checked, for a sample of 5 random attributes of the selected ones, that all the values not available in the JSON files were also not available in the HTML source of the corresponding page. Overall, out of the initial 587 attributes available in the Alaska benchmark, our data extraction ground-truth includes 92 attributes most of which are published only by a minority of the sites in the dataset.

7.2.2 Results on the Alaska Benchmark. The experimental evaluation on the Alaska benchmark is carried out by using the optimal algorithm A*, and the two Greedy variants to benchmark and analyse the amount of backtracking needed to get the optimal solution.

We used 10% of the pages of each site as training set.[8] The results shown in Figure 6 feature an average precision and recall of 78.15% and 84.20%, respectively. Figure 6a also shows the percentage of additional splits that A* needed for every site in the Alaska benchmark (Δ Split). On average Greedy saves 21.3% of splits (with a standard deviation of 8.23%).

Figure 6c plots the differences between the number of splits operated by the A* and Greedy algorithms w.r.t. the size of the regions involved by the split. The split is counted once for every involved region. This experiment shows that the vast majority of additional splits operated by A* while backtracking is for small regions with less than 4-5 tokens.

7.2.3 The parameter k. To evaluate the impact of the k parameter in the search space for solving k-SEP, we illustrate the results of a few experiments conducted on some representative websites from the Alaska benchmark by averaging over all the attributes as k ranges from 2 to 12. The continuous lines in Figure 6d (left vertical axis) shows the F-measure for the wrapper produced by the A* algorithm (similar results are obtained with Greedy). The larger the value of k, the fewer fine-grained details of the template can be observed: on loosely templated websites (e.g., buzzillions.com and cambuy.com.au) the F-measure significantly drops, as soon as some optional portion of the HTML template occurring less than k times in the sample pages cannot be properly identified as such anymore. On the contrary, the template of regularly structured websites (e.g., gosale.com and pcconnection.com) are still perfectly observable and the F-measure is steady even for large values of k.

Figure 6d also plots with dashed lines (right vertical axis — Δ Split) the percentage of splits which are saved as k increases. The reference is the number of splits performed for $k = 2$. It can be noticed that the two sites with the largest savings are those that also experienced a significant F-measure drop: the additional splits are used for modelling fine-grained details of the template.

7.2.4 Experiments with Undersampled Input. In many practical scenarios, the quality of the training set used to infer a wrapper is not known up-front. A comprehensive evaluation of an unsupervised generator should also consider the quality of the output wrapper when the set of training pages is not large enough to provide all the needed statistical evidence of every fine-grained variation of the underlying HTML template.[9]

We simulate these scenarios by means of undersampled collections of pages from the Alaska benchmark. The plots depicted in Figures 6e with a continuous line show the average F-measure (left vertical axis) for A* and Greedy. The results are averaged over all the attributes as the percentage of training pages increases from 2% to 10% of the dataset size. The A* algorithm achieves an average 59.35% precision and 58.80% recall with a training set of only 4% pages. Interestingly, Greedy-inf performs more often closer to A* than to Greedy-rnd: the policy to break ties has a direct impact on the learning algorithm in these undersampled scenarios.

7.2.5 HTML Dynamic Tokenization. In order to evaluate the contribution of the dynamic tokenization technique presented in Section 6, we ran again the experiments described in Section 7.2.4 with dynamic tokenization disabled. The dashed lines in Figures 6e plot the differences between the F-measure of the output wrapper obtained in the two experiments (right vertical axis — ΔF-measure), i.e., the improvements due to the technique.

We intentionally consider only settings in which the number of training pages is rather small (less than 10%) so that the statistical evidence to classify tokens as either template or content is scarce. With dynamic tokenization disabled, there is a drop both in terms of precision and of recall, due to the reduced ability to properly distinguish tags having the same element name but different roles in the underling HTML template. The smaller the percentage of pages used to induce a wrapper, the more significant its contribution to A*'s performance. On the contrary, since Greedy does not backtrack, it cannot fully exploit dynamic tokenization: its contribution to Greedy's performance becomes quickly negligible as the size of the training set gets smaller. To conclude, this experiment confirms that the advantages of A* over Greedy are more remarkable when the input collection of pages is undersampled and dynamic tokenization is enabled.

8 RELATED WORKS

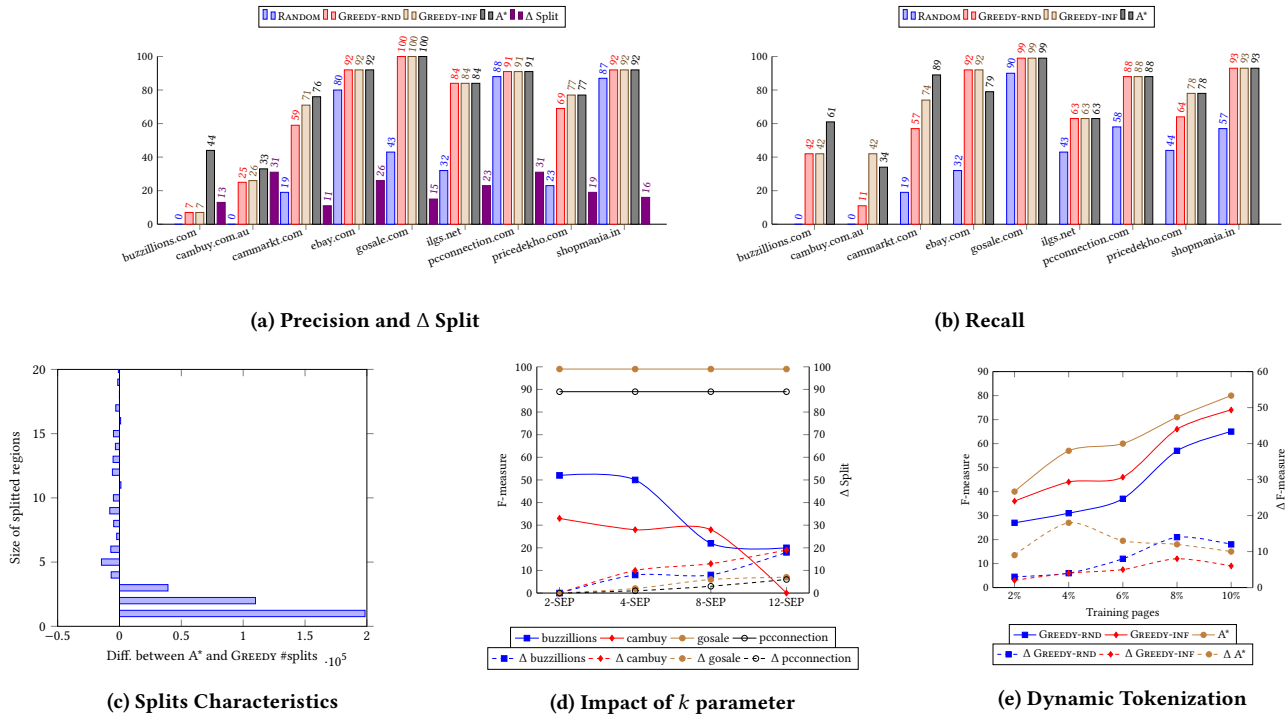
Extracting data from pages is a long-standing and challenging research and industrial problem on which at least twenty years of research [59] have been spent. Several startups have been created with the explicit goal of extracting and leveraging Web data, such as Lixto [45], Wrapidity [47], Diffbot [43], and import.io [44]. It also well known that many companies, of every size, create and maintain for their business lines ad-hoc solutions to gather and process Web data, sometimes even at massive scale [46].

Vertex [28] has been one of the first documented large scale end-to-end production system that solves several tasks (including grouping pages by structure, and learning XPath-based rules that are robust to page changes). It is based on a supervised approach in which human operators have to provide a few annotated pages.

In scientific research, several trends can be highlighted: as for the level of automation, the proposed techniques initially focused on increasing the level of automation of the inference process,

[8] Except for ebay.com site that contains ~7.1k pages, for which we use 70 pages.

[9] It also a quite common practice to collect pages by means of inherently biased crawling programs, such as those collecting items from top-lists.



(a) Precision and Δ Split **(b) Recall**

(c) Splits Characteristics **(d) Impact of k parameter** **(e) Dynamic Tokenization**

Figure 6: Experiments on the Alaska Benchmark

ranging from ad hoc formal languages for writing wrapper [11], to supervised solution [1, 25, 28, 35, 58], and also fully automatic solutions [2, 14]; as for the scale, the initial solutions aim at extracting data from a single site [1], whereas later approaches were designed to deal with many websites [15, 22, 29]; as for the kind of patterns exploited, regular expressions over HTML-syntax based tokens [2, 13], visual features [36, 60, 61], and several types of annotations [22, 38, 41, 50], have all been covered, sometimes by also exploiting the redundancy across Web sources [6, 37]; as for the type and variety of sources, a recent trend is that of proposing solutions able to gather data holistically not only from templated websites, but also from other type of sources [19, 38, 39].

Unsupervised approaches like RoadRunner [12], ExAlg [2], FiVaTech [33], Trinity [56], and ViDE [36] are all based on the analysis of the differences between pages generated by using the same HTML template. It is then possible to reverse engineer a model of that template, for example by means of regular expressions [2], or one of their union-free subsets [12]. However, in practice, all these systems are not used for large scale extraction tasks: their performances are too dependent on how well the input pages satisfy the underlying assumptions. The generated wrapper turns out to be extremely sensitive to even small details in the pages of the training set, and their quality is highly unpredictable at scale [16].

In sharp contrast to many of the previous approaches that struggled to infer a token-level description of each and every detail of a HTML template, the local parsability property of landmark grammars allows the generation of wrappers that can extract data by confining the effect of parsing failures even in presence of unexpected variations of the underlying template in the parsed pages.

The concept of token *role* was first mentioned in [2] together with other techniques for distinguishing token occurrences (e.g., associating each one with its canonical-XPath, e.g., `/html[1]/body[1]/b[2]`): we took inspiration from this technique to clarify the challenges posed by HTML ambiguity. However, we get its consequences to the fullest extent (see Section 6).

The local parsability property can help to tackle the same challenges faced by the approaches that discussed the robustness of the wrappers such as [9, 18, 23, 48]. They introduced techniques to infer robust, *over time*, XPath expressions for extracting a single annotated attribute. In contrast, our technique focuses on a non-directional parsing algorithm that allows to confine the effect of any type misalignment/error, independently of its nature and origin. The property has been introduced in the context of Floyd grammars [21, 26] to handle parsing errors [17, 51].

A problem originally formulated as: "*What is the smallest context-free grammar that generates exactly one given string?*", titles a paper [8] in the early 2000s, but the same problem was already posed by several authors in the second half of 1990s, as one of the basic tools used for developing loss-less data compression solutions [34], and can be traced back, in different formulations [7] to older literature [57]. A recent proposal [5] shifts from the original formulation with one input string [40] to a generalized version in which several strings must be parsed by the output grammar [55].

REFERENCES

[1] Brad Adelberg. 1998. NoDoSE—a tool for semi-automatically extracting structured and semistructured data from text documents. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*. 283–294.

[2] Arvind Arasu and Hector Garcia-Molina. 2003. Extracting structured data from web pages. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. ACM, 337–348.

[3] Jean-Christophe Aval. 2008. Multivariate fuss–catalan numbers. *Discrete Mathematics* 308, 20 (2008), 4660–4669.

[4] Mohd Amir Bin Mohd Azir and Kamsuriah Binti Ahmad. 2017. Wrapper approaches for web data extraction: A review. In *2017 6th International Conference on Electrical Engineering and Informatics (ICEEI)*. IEEE, 1–6.

[5] Hideo Bannai, Momoko Hirayama, Danny Hucke, Shunsuke Inenaga, Artur Jez, Markus Lohrey, and Carl Philipp Reh. 2019. The smallest grammar problem revisited. *CoRR* abs/1908.06428 (2019). arXiv:1908.06428 http://arxiv.org/abs/1908.06428

[6] Mirko Bronzi, Valter Crescenzi, Paolo Merialdo, and Paolo Papotti. 2013. Extraction and Integration of Partially Overlapping Web Sources. *Proc. VLDB Endow.* 6, 10 (2013), 805–816. https://doi.org/10.14778/2536206.2536209

[7] Katrin Casel, Henning Fernau, Serge Gaspers, Benjamin Gras, and Markus Schmid. 2020. On the Complexity of the Smallest Grammar Problem over Fixed Alphabets. *Theory of Computing Systems* (11 2020), 1–66. https://doi.org/10.1007/s00224-020-10013-w

[8] M. Charikar, E. Lehman, Ding Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. 2005. The smallest grammar problem. *IEEE Transactions on Information Theory* 51, 7 (2005), 2554–2576. https://doi.org/10.1109/TIT.2005.850116

[9] Boris Chidlovskii, Bruno Roustant, and Marc Brette. 2006. Documentum ECI Self-Repairing Wrappers: Performance Analysis. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data* (Chicago, IL, USA) *(SIGMOD '06)*. Association for Computing Machinery, New York, NY, USA, 708–717. https://doi.org/10.1145/1142473.1142555

[10] Valter Crescenzi, Andrea De Angelis, Donatella Firmani, Maurizio Mazzei, Paolo Merialdo, Federico Piai, and Divesh Srivastava. 2021. Alaska: A Flexible Benchmark for Data Integration Tasks. arXiv:2101.11259 [cs.DB]

[11] Valter Crescenzi and Giansalvatore Mecca. 1998. Grammars have exceptions. *Information Systems* 23, 8 (1998), 539–565.

[12] Valter Crescenzi and Giansalvatore Mecca. 2004. Automatic information extraction from large websites. *Journal of the ACM (JACM)* 51, 5 (2004), 731–779.

[13] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. 2004. Handling irregularities in roadrunner. In *AAAI-04 ATEM Workshop*.

[14] Valter Crescenzi, Giansalvatore Mecca, Paolo Merialdo, et al. 2001. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB*, Vol. 1. 109–118.

[15] Valter Crescenzi, Paolo Merialdo, and Disheng Qiu. 2015. Crowdsourcing large scale wrapper inference. *Distributed and Parallel Databases* 33, 1 (2015), 95–122.

[16] Valter Crescenzi, Paolo Merialdo, and Disheng Qiu. 2019. Hybrid Crowd-Machine Wrapper Inference. *ACM Trans. Knowl. Discov. Data* 13, 5, Article 51 (Sept. 2019), 43 pages. https://doi.org/10.1145/3344720

[17] Stefano Crespi Reghizzi, Violetta Lonati, Dino Mandrioli, and Matteo Pradella. 2017. Toward a Theory of Input-Driven Locally Parsable Languages. *Theor. Comput. Sci.* 658, PA (Jan. 2017), 105–121. https://doi.org/10.1016/j.tcs.2016.05.003

[18] Nilesh Dalvi, Philip Bohannon, and Fei Sha. 2009. Robust web extraction: an approach based on a probabilistic tree-edit model. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. ACM, 335–348.

[19] Xin Luna Dong, Xiang He, Andrey Kan, Xian Li, Yan Liang, Jun Ma, Yifan Ethan Xu, Chenwei Zhang, Tong Zhao, Gabriel Blanco Saldana, et al. 2020. AutoKnow: Self-driving knowledge collection for products of thousands of types. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2724–2734.

[20] Steve Faulkner, Arron Eicholz, Travis Leithead, Alex Danilo, and Sangwhan Moon. 2017. HTML 5.2. *W3C. Retrieved January* 17 (2017), 2018.

[21] Robert W Floyd. 1963. Syntactic analysis and operator precedence. *Journal of the ACM (JACM)* 10, 3 (1963), 316–333.

[22] Tim Furche, Georg Gottlob, Giovanni Grasso, Xiaonan Guo, Giorgio Orsi, Christian Schallhart, and Cheng Wang. [n.d.]. DIADEM: Thousands of Websites to a single database. PVLDB 7 (14), 1845–1856 (2014).

[23] Tim Furche, Jinsong Guo, Sebastian Maneth, and Christian Schallhart. 2016. Robust and Noise Resistant Wrapper Induction. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA) *(SIGMOD '16)*. Association for Computing Machinery, New York, NY, USA, 773–784. https://doi.org/10.1145/2882903.2915214

[24] David Gibson, Kunal Punera, and Andrew Tomkins. 2005. The volume and evolution of web page templates. In *Special interest tracks and posters of the 14th international conference on World Wide Web*. 830–839.

[25] Georg Gottlob, Christoph Koch, Robert Baumgartner, Marcus Herzog, and Sergio Flesca. 2004. The Lixto data extraction project: back and forth between theory and practice. In *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 1–12.

[26] Dick Grune. 2010. *Parsing Techniques: A Practical Guide* (2nd ed.). Springer Publishing Company, Incorporated.

[27] Dick Grune and Ceriel JH Jacobs. 2007. Parsing Techniques. *Monographs in Computer Science. Springer,* (2007), 13.

[28] Pankaj Gulhane, Amit Madaan, Rupesh Mehta, Jeyashankher Ramamirtham, Rajeev Rastogi, Sandeep Satpal, Srinivasan H Sengamedu, Ashwin Tengli, and Charu Tiwari. 2011. Web-scale information extraction with vertex. In *2011 IEEE 27th International Conference on Data Engineering*. 1209–1220. https://doi.org/10.1109/ICDE.2011.5767842

[29] Jinsong Guo, Valter Crescenzi, Tim Furche, Giovanni Grasso, and Georg Gottlob. 2019. RED: Redundancy-Driven Data Extraction from Result Pages?. In *The World Wide Web Conference*. ACM, 605–615.

[30] Qiang Hao, Rui Cai, Yanwei Pang, and Lei Zhang. 2011. From one tree to a forest: a unified solution for structured web data extraction. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*. 775–784.

[31] Bin He, Mitesh Patel, Zhen Zhang, and Kevin Chen-Chuan Chang. 2007. Accessing the deep web. *Commun. ACM* 50, 5 (2007), 94–101.

[32] Crunchbase Inc. 2013. Lixto acquired by McKinsey. https://www.crunchbase.com/organization/lixto-software.

[33] Mohammed Kayed and Chia-Hui Chang. 2010. FiVaTech: Page-Level Web Data Extraction from Template Pages. *IEEE Transactions on Knowledge and Data Engineering* 22, 2 (2010), 249–263. https://doi.org/10.1109/TKDE.2009.82

[34] J. C. Kieffer and En-Hui Yang. 2006. Grammar-Based Codes: A New Class of Universal Lossless Source Codes. *IEEE Trans. Inf. Theor.* 46, 3 (Sept. 2006), 737–754. https://doi.org/10.1109/18.841160

[35] Alberto HF Laender, Berthier A Ribeiro-Neto, Altigran S Da Silva, and Juliana S Teixeira. 2002. A brief survey of web data extraction tools. *ACM Sigmod Record* 31, 2 (2002), 84–93.

[36] Wei Liu, Xiaofeng Meng, and Weiyi Meng. 2010. ViDE: A Vision-Based Approach for Deep Web Data Extraction. *IEEE Transactions on Knowledge and Data Engineering* 22, 3 (2010), 447–460. https://doi.org/10.1109/TKDE.2009.109

[37] Colin Lockard, Xin Luna Dong, Arash Einolghozati, and Prashant Shiralkar. 2018. Ceres: Distantly supervised relation extraction from the semi-structured web. *arXiv preprint arXiv:1804.04635* (2018).

[38] Colin Lockard, Prashant Shiralkar, and Xin Luna Dong. 2019. OpenCeres: When open information extraction meets the semi-structured web. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 3047–3056.

[39] Colin Lockard, Prashant Shiralkar, Xin Luna Dong, and Hannaneh Hajishirzi. 2020. Web-scale Knowledge Collection. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 888–889.

[40] Markus Lohrey. 2012. Algorithmics on SLP-compressed strings: A survey. *Groups Complexity Cryptology* 4, 2 (2012), 241–299. http://dblp.uni-trier.de/db/journals/gcc/gcc4.html#Lohrey12

[41] Stefano Ortona, Giorgio Orsi, Marcello Buoncristiano, and Tim Furche. 2015. WADaR: Joint wrapper and data repair. *Proceedings of the VLDB Endowment* 8, 12 (2015), 1996–1999.

[42] Stefano Ortona, Giorgio Orsi, Tim Furche, and Marcello Buoncristiano. 2016. Joint repairs for web wrappers. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. IEEE, 1146–1157.

[43] Home page. 2021. Diffbot. https://www.diffbot.com/.

[44] Home page. 2021. import.io. https://www.import.io/.

[45] Home page. 2021. Lixto. http://www.lixto.com/.

[46] Home page. 2021. Meltwater: Media Monitoring & Social Listening Platform. https://www.meltwater.com/.

[47] Home page. 2021. Wrapidity. https://www.wrapidity.com.

[48] Aditya Parameswaran, Nilesh Dalvi, Hector Garcia-Molina, and Rajeev Rastogi. 2011. Optimal Schemes for Robust Web Extraction. *Proceedings of the VLDB Conference* 4, 11 (September 2011). http://ilpubs.stanford.edu:8090/998/

[49] Judea Pearl. 1984. Heuristics: intelligent search strategies for computer problem solving. (1984).

[50] Mohammad Raza and Sumit Gulwani. 2020. Web Data Extraction using Hybrid Program Synthesis: A Combination of Top-down and Bottom-up Inference. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1967–1978.

[51] Stefano Crespi Reghizzi, Luca Breveglieri, and Angelo Morzenti. 2013. *Formal languages and compilation*. Springer.

[52] Stefano Crespi Reghizzi, Luca Breveglieri, and Angelo Morzenti. 2019. *Formal languages and compilation*. Springer.

[53] Stefano Crespi Reghizzi and Dino Mandrioli. 2012. Operator precedence and the visibly pushdown property. *J. Comput. System Sci.* 78, 6 (2012), 1837–1867.

[54] Stuart Russel, Peter Norvig, et al. 2013. *Artificial intelligence: a modern approach*. Pearson Education Limited.

[55] Payam Siyari and Matthias Gallé. 2017. The Generalized Smallest Grammar Problem *(Proceedings of Machine Learning Research)*, Sicco Verwer, Menno van Zaanen, and Rick Smetsers (Eds.), Vol. 57. PMLR, Delft, The Netherlands, 79–92.

http://proceedings.mlr.press/v57/siyari16.html

[56] Hassan A. Sleiman and Rafael Corchuelo. 2014. Trinity: On Using Trinary Trees for Unsupervised Web Data Extraction. *IEEE Transactions on Knowledge and Data Engineering* 26, 6 (2014), 1544–1556. https://doi.org/10.1109/TKDE.2013.161

[57] James A. Storer and Thomas G. Szymanski. 1982. Data Compression via Textual Substitution. *J. ACM* 29, 4 (Oct. 1982), 928–951. https://doi.org/10.1145/322344.322346

[58] Fergus Toolan and Nicholas Kusmerick. 2002. Mining web logs for personalized site maps. In *Proceedings of the Third International Conference on Web Information Systems Engineering (Workshops), 2002.* Citeseer, 232–237.

[59] Xiaoying Wu and Dimitri Theodoratos. 2013. A survey on XML streaming evaluation techniques. *The VLDB Journal* 22, 2 (2013), 177–202.

[60] Yanhong Zhai and Bing Liu. 2005. Web data extraction based on partial tree alignment. In *Proceedings of the 14th international conference on World Wide Web*. 76–85.

[61] Hongkun Zhao, Weiyi Meng, Zonghuan Wu, Vijay Raghavan, and Clement Yu. 2005. Fully automatic wrapper generation for search engines. In *Proceedings of the 14th international conference on World Wide Web*. 66–75.