RED: Redundancy-Driven Data Extraction from Result Pages*

Jinsong Guo
University of Oxford
United Kingdom
jinsong.guo@cs.ox.ac.uk

Valter Crescenzi
University Roma Tre
Italy
crescenz@dia.uniroma3.it

Tim Furche
University of Oxford & Meltwater
United Kingdom
tim.furche@cs.ox.ac.uk

Giovanni Grasso
University of Calabria & Meltwater
Italy
giovanni.grasso@unical.it

George Gottlob
University of Oxford & TU Wien
United Kingdom
georg.gottlob@cs.ox.ac.uk

ABSTRACT

Data-driven websites are mostly accessed through search interfaces. Such sites follow a common publishing pattern that, surprisingly, has not been fully exploited for unsupervised data extraction yet: the result of a search is presented as a paginated list of result records. Each result record contains the main attributes about one single object, and links to a page dedicated to the details of that object.

We present RED, an automatic approach and a prototype system for extracting data records from sites following this publishing pattern. RED leverages the inherent redundancy between result records and corresponding detail pages to design an effective, yet fully-unsupervised and domain-independent method. It is able to extract from result pages all the attributes of the objects that appear both in the result records and in the corresponding detail pages.

With respect to previous unsupervised methods, our method does not require any a priori domain-dependent knowledge (e.g, an ontology), can achieve a significantly higher accuracy while automatically selecting only object attributes, a task which is out of the scope of traditional fully unsupervised approaches. With respect to previous supervised or semi-supervised methods, RED can reach similar accuracy in many domains (e.g., job postings) without requiring supervision for each domain, let alone each website.

CCS CONCEPTS

• **Information systems → Data extraction and integration**.

KEYWORDS

Web Data Extraction; Automatic Wrapper Generation; XPath

ACM Reference Format:
Jinsong Guo, Valter Crescenzi, Tim Furche, Giovanni Grasso, and George Gottlob. 2019. RED: Redundancy-Driven Data Extraction from Result Pages. In *Proceedings of the 2019 World Wide Web Conference (WWW'19), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3308558.3313529

*This work is supported by the EPSRC programme grant EP/M025268/1 VADA, and the EU Horizon 2020 grant 809965 LAMBDA.

WWW '19, May 13–17, 2019, San Francisco, CA, USA

ACM ISBN 978-1-4503-6674-8/19/05.
https://doi.org/10.1145/3308558.3313529

1 INTRODUCTION AND OVERVIEW

Data-driven websites across many application domains (e.g., real estate agencies, e-commerce) follow a common publishing pattern to provide access to the underlying data: A search interface to query specific types of objects (e.g., products, properties) yields the relevant objects as a list of result records, often paginated into *result pages*. Every result record contains key attributes about one single object, so that the user can get a first impression of the result before deciding to dig further into detailed information. Each result record includes a link to a *detail page*, containing more attributes [24].

To the best of our knowledge, no existing data extraction system has targeted the inherent data intra-site redundancy underlying this popular publishing pattern. Rather, most existing unsupervised approaches [2, 8, 9, 29, 30, 34, 35] can be applied either over a collection of result pages, or over a collection of detail pages. Other approaches rely on the same publishing pattern, but focus only on segmenting the result pages [28], or rely on the much weaker signals arising from the aligning of the labels of the fields on the search form directly against the labels of the data on the detail pages [37]; finally, there are several approaches that focus on the problem of finding redundancy [3, 38] among several sites, but the problem quickly trespasses on that of integrating data coming from autonomous sources [3, 5, 6, 23, 38], a problem that is well known not to have a simple solution [14].

This paper proposes RED, the first data extraction method that leverages this intra-site redundancy to extract data records from result pages. Given a set of result pages and the corresponding set of detail pages from a website, RED infers a collection of *extraction rules* (or simply *rules*). Each one is capable of extracting the values for a certain attribute, e.g., the price of a product, from all the records of a result page. These rules can be applied to every result page of the same website to extract all the relevant data.

We show that exploiting this publishing pattern on a site provides a *low-hanging fruit* opportunity for significantly improving the precision of fully unsupervised data extraction. Existing unsupervised approaches struggle to distinguish relevant data (i.e., object attributes) from noise. Moreover, they are based on a fragile trade-off between the expressiveness of formalism used for describing the extraction rules and the efficiency of the learning task. Conversely, intra-site redundancy distinguishes relevant data that appears both on result records and on detail pages from noise that only appears on either without any supervision, and allows to filter

out noisy candidate extraction rules even when using an expressive formalism to specify them.

Key idea: Exploit the inherent intra-site redundancy between result records and their corresponding detail pages to fully automatically distinguish between slots in the template structure that correspond to actual object attributes.

Figure 1 shows a (simplified) instance of a real estate website that serves as running example. It consists of a result page containing six records along with their corresponding detail pages. The result records and detail pages publish attributes such as Price, Location, number of Beds, and Type of properties. Colored annotations highlight overlapping values for attributes appearing within the records in the result page and the corresponding detail pages. For this example, RED is capable of generating rules extracting the correct values of all attributes for all result records. Figure 2a shows the values extracted by the extraction rules when applied to the example result page in input.

RED generates these rules in three, sequential steps: **(1)** During the *extraction rules generation* step, RED generates two sets of rules, namely the sets of *result rules* and *detail rules*: Figure 2b and Figure 2c show the values extracted by some of the generated rules (shown in Figure 5a and Figure 5b) on the result page and on the detail pages, respectively, for our running example (where r_i (d_i) is the i-th result (detail) rule generated). RED's extraction rules generation algorithm (Section 3) selects the rules from a fragment of XPath [13] expressions that, as empirically verified over many real websites, includes correct extraction rules.

The main challenge in this step is generating a *complete* set of rules efficiently: at least one correct detail rule and one correct result rule is needed for every attribute. It turns out that the completeness of the rules generation algorithm can be achieved only at expense of using a rather expressive family of extraction rules, which means generating many candidate rules, most of which are actually selecting noise.

(2) In the second step, *redundancy seeking*, RED aims to identify pairs of result/detail rules that are considered *redundant*, as they extract the same values for every object. In this quest for redundancy, RED is faced with various challenges. First, in addition to *neat* redundancy between correct rules, there is inevitably a lot of *noisy* redundancy: Some values may occur multiple times across the records (typical if the search is narrow, e.g., only properties in London, or because attributes have a narrow domain, e.g., Beds). Moreover, some values may be repeated several times within the same record or detail page (the third detail page contains a second occurrence of '`Studio`', dotted-blue underlined in Figure 1).

To complicate matters, certain attributes may be optional i.e., not report a value for some result record, such as the Price attribute which is missing for the third and sixth properties of our running example. Both rules r_0 and d_0 select Price values: d_0 is applied to each of the detail pages separately, and can therefore extract the price value if occurring on the page, or it can just report that the value is missing (indicated by *nil*). Conversely, r_0 is applied once on the whole result page (and not on the individual records): it can only extract the price values found on the page, irrespective of which record they belong to. Therefore, taken alone, r_0 is not able to distinguish which records contain a value, and which do not (in our example the third and last record). That leaves r_0 and d_0 mis-"aligned" and without further consideration they would not be considered redundant. This problem has been already tackled by known complex and error prone segmentation techniques for splitting a result page into records [19, 28, 29]. Rather, RED devises an innovative soft-segmentation technique (Section 4.1) which, again, leverages the underlying intra-site redundancy: In particular, RED exploits the presence of navigational links (named *detail links*) that point from each result record to the corresponding detail page. In Figure 1, for each record, one occurrence of the detail link is highlighted (by the departing orange solid arrow). All these detail link occurrences are captured by an extraction rule l, named *link rule*, whose generation is facilitated by the knowledge of urls of the detail pages given in input to RED.

(3) In its final step, named *noise removal*, RED addresses the issue of finding and discarding pairs of rules that exhibit some limited redundancy, but not sufficient to likely be correct extraction rules. RED addresses this challenge by implementing a noise cleaning process (Section 5) that aims at separating the redundancy resulting from the result/detail publishing pattern, from that arising by accident.

Finally, RED outputs all the result extraction rules left after the previous steps along with the special link rule, as shown in Figure 2a for the running example. It is worth noting that, while in this paper we are focusing on data extraction from result page records, RED is perfectly capable of inferring valid extraction rules also for attributes published on the detail pages. Aside of space reasons, we focus on result rules as executing only such rules at a large scale is much cheaper: Once the rules have been generated, no more detail pages have to be crawled, and that is a significant saving, as there are many more detail pages than result pages.

The paper is organized around the main contributions of RED:

- The first formal description of the typical result/detail publishing pattern and the definition of the *Finding Result Attributes Problem* (Section 2);
- A highly accurate, domain independent, and fully unsupervised data extraction method that leverages the redundancy implied by the publishing pattern on real-life websites. It is presented in three steps: a rules generation algorithm to efficiently generate complete sets of extraction rules (Section 3); a technique to measure the redundancy of rules (Section 4), even in the presence of optional attributes (Section 4.1); and finally, a technique to separate and validate regular, attribute redundancy from noisy, accidental redundancy (Section 5);
- A proof-of-concept showing the feasibility of the approach, and the experimental evaluation (Section 6) showing that RED can attain highly accurate results scoring at rule level an F-measure $F_r > 91\%$, and even better results at value level with $F_v > 96\%$. We demonstrate its advantages against state-of-the-art large scale data extraction systems.

The following related problems have been already tackled in literature and are beyond the scope of the present paper: finding deep Web sources [36, 39, 40, 43]; filling the search fields with meaningful values to collect result pages [1, 4, 17, 25, 31–33, 42]; crawling paginated search result pages [21].

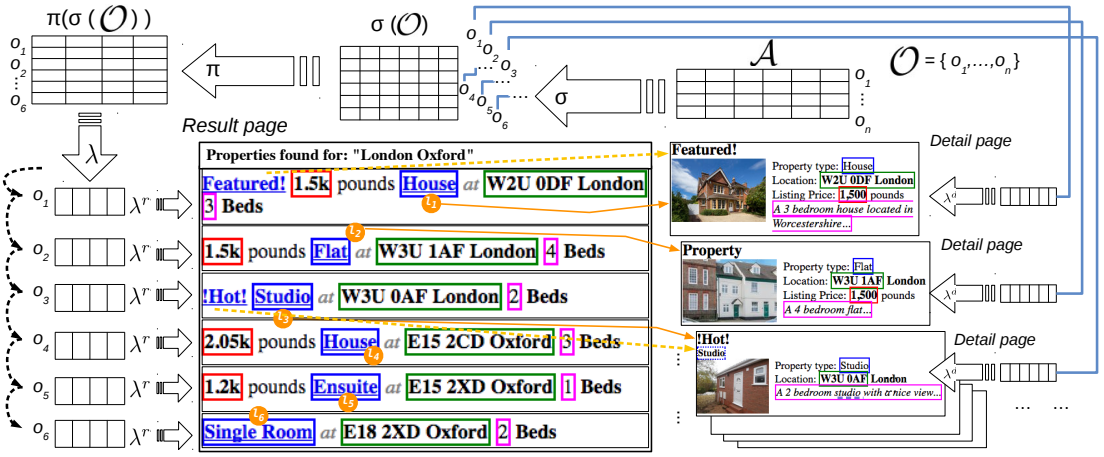


Figure 1: The result/detail publishing pattern.

r_0^b (Price)	r_2 (Type)	r_6 (Location)	r_9 (Beds)	l (Link)
1.5k	House	W2U 0DF London	3	l_1
1.5k	Flat	W3U 1AF London	4	l_2
nil	Studio	W3U 0AF London	2	l_3
2.05k	House	E15 2CD Oxford	3	l_4
1.2k	Ensuite	E15 2XD Oxford	1	l_5
nil	Single Room	E18 2XD Oxford	2	l_6

(a) RED's Output on the Running Example: Annotated rules to extract correct result records.

r_0: ⟨1.5k, 1.5k, 2.05k, 1.2k⟩
r_1: ⟨1.5k, Flat, Studio, House, Ensuite, at⟩
r_2: ⟨House, Flat, Studio, House, Ensuite, Single Room⟩
r_3: ⟨Featured!, 1.5k, 2.05k, 1.2k⟩
r_4: ⟨1.5k, 1.5k, 2.05k, 1.2k⟩
r_5: ⟨at, W3U 1AF Lon..., W3U 0AF Lon..., E15 2CD Ox..., E15 2XD Ox...⟩
r_6: ⟨W2U 0DF Lon..., W3U 1AF Lon..., ...,..., E15 2XD Ox..., E18 2XD Ox...⟩
r_7: ⟨at, W3U 1AF Lon..., W3U 0AF Lon..., E15 2CD Ox..., E15 2XD Ox...⟩
r_8: ⟨W2U 0DF Lon..., W3U 1AF Lon..., E15 2CD Ox..., E15 2XD Ox...⟩
r_9: ⟨3, 4, 2, 3, 1, 2⟩

(b) Values extracted by result rules.

d_0: ⟨1,500, 1,500, *nil*, 2,050, 1,200, *nil*⟩
d_1: ⟨House, Flat, Studio, House, Ensuite, Single Room⟩
d_2: ⟨Featured!, Property, !Hot!, Property, Property, Property⟩
d_3: ⟨pounds, pounds, A 2 bedroom studio..., pounds, pounds, A single...⟩
d_4: ⟨W2U 0DF Lon..., W3U 1AF, W3U 0AF, E15 2CD Ox..., E15 2XD, E18...⟩
d_5: ⟨W2U 0DF Lon..., W3U 1AF, *nil*, E15 2CD Ox..., E15 2XD, *nil*⟩
d_6: ⟨A 3 bedroom house..., A 4 bedroom flat..., A 2 bedroom studio..., ···⟩
d_7: ⟨*nil*, London, London, *nil*, Oxford, Oxford⟩

(c) Values extracted by detail rules.

Figure 2: Running example.

2 PROBLEM DESCRIPTION

The result/detail publishing pattern depicted in Figure 1 assumes that a website publishes data coming from an underlying abstract relation with attributes $\mathcal{A}$ containing all the available information about a set of objects O. In our example, the abstract relation contains one tuple for every published property, each with attributes such as the property's Price, Address, number of Beds, and Type.

Result pages and detail pages are generated by various *scripts* that retrieve data from the abstract relation, embed them into an HTML template, and publish them as web pages. This process performs a sequence of transformations: An initial selection (σ, to recycle a symbol from relational algebra) produces the set of objects to publish in each result page as encoded into HTML source code by the *result page script* λ. Every result page contains a set of *result records*, each produced by applying the *result record script* λ^r to one published object. The corresponding detail page is the result of applying the *detail script* λ^d on the very same object, and is linked from the corresponding result record.

According to the model depicted in Figure 1, even if the detail and result record scripts work on the same set of objects, they may end up publishing different sets of attributes. The detail pages include all the attributes of the abstract relation, whereas the result pages publish only a subset of these attributes selected by a projection operation π.

We now introduce the problem of recovering the values of the attributes published on the result pages. An extraction rule either locates one or several string values from a single page, or it produces a distinct special value *nil* to denote the absence of a value. For our purposes, extraction rules are specified by using XPath expressions [13] belonging to a simple but carefully designed fragment (as detailed in Section 3).

We distinguish two types of extraction rules, named *detail* and *result* rules after the type of pages they are meant to work on.

We call *detail rule* any extraction rule that when applied on a detail page, produces at most one value or *nil*. We use d to denote a detail extraction rule, and we use $d(p)$ to denote the value it extracts from detail page p. Precisely, $d(p)$ is either the XPath *string-value* obtained by applying the rule d on detail page p, or it is *nil* if the XPath expression returns an empty *node-set* [13].

A detail rule for an attribute A is said to be *correct* if it extracts its values from every detail page in the input and extracts *nil* for those pages that do not contain a value for A. We call *noisy* (or *incorrect*) a rule that is not correct: noisy rules mix values of several attributes or they extract part of the underlying HTML template. We distinguish the special case of a *partially-correct* rule for an attribute A, i.e., a rule extracting only correct values of attribute A except for some pages, on which it wrongly extracts *nil*. The value extracted by a detail rule is naturally associated with the object corresponding to the detail page it is applied to.

Similarly, we call *result rule* any extraction rule meant to work on result pages and we denote it by r. A result rule applied on a result page p produces zero, one, or several values denoted $r(p)$. Precisely, r returns the *string-value* for each node in the *node-set* returned by evaluating its XPath expression on the result page p [13].

Result rules, differently from detail rules, cannot produce *nil* values at all, even in the case that they return an empty node-set. Indeed, without knowing the exact boundaries of every result record, the values extracted by a result rule cannot be trivially associated to the record/object they belong to on the result page (Section 4.1 describes a solution to this problem).

Therefore, the notion of *correctness* of a result rule for an attribute A differs from the notion of correctness of a detail rule, even if they are associated with the same attribute: a result rule is correct if it produces exactly the ordered set of non-*nil* values of attribute A as they occur in the source of that page. By *partially-correct* result rule for an attribute A we mean a result rule extracting a strict subset of the correct values of A.

The problem that we aim to solve can be formulated as follows:

Problem (Finding Result Attributes). *Given a set of result pages P_r and the set of corresponding detail pages P_d over the same objects, find the correct values (including nil values) for every attribute A of the abstract relation published in the result pages.*

RED tries to solve this problem by producing a result rule r_A per every attribute A in the result pages, together with an additional result rule, named *link rule* and denoted by l, extracting exactly one occurrence of the link leading to the corresponding detail page per result record: an output result rule r_A associated with an optional attribute A is suitably annotated either r_A^a or r_A^b to specify how the values extracted by r_A should be padded with missing *nil* values, i.e., respectively *after* or *before* occurrences of the detail links.

3 EXTRACTION RULES GENERATION

We introduce an extraction rules generation algorithm working on both result and detail pages. It is a single-parameter algorithm designed to output rules from an XPath fragment whose expressiveness can be easily and effectively tuned by setting the parameter. Its goals are two-fold: on one hand, it aims at being complete, i.e., it has to generate at least a correct rule for every attribute; on the other hand, it should not generate too many noisy rules, because their presence makes the *Finding Result Attributes* problem harder.

The rules generation algorithm includes two main steps: *template analysis*, and *extraction rules enumeration*.

3.1 Template Analysis

The template analysis aims at identifying those nodes in the DOM tree [26] of the input pages that are *template* nodes (e.g., the `<li>` node, and the textual label: '`Price:`' in Figure 3a).[1]

RED's template analysis algorithm is inspired by ExAlg [2], suitably adapted to the result/detail publishing pattern. Our analysis just aims at deciding, for every node in the pages, whether it should be considered as part of the template or not; conversely, ExAlg also solves the complex problem of finding a *full* description of the HTML template.

The analysis on a set of input pages P builds on the notion of *occurrence-vector*, i.e., a vector f of $|P|$ integers indexed by the pages in P, so that $f(p)$ reports how many occurrences of *equivalent* nodes are present in the DOM tree of page p. Two nodes are said to be equivalent if and only if two conditions hold: (i) they are either text nodes with the same value, or they both are element nodes associated with the same element name and attribute names, (ii) their respective parent nodes are equivalent (or they both are the root nodes). Equivalent nodes with the same occurrence-vector are then grouped into the same *equivalence class*.

As observed in [2], by considering a sufficiently large number of pages, the nodes that occur in large and frequently occurring classes are the scaffold of the underlying template. An inherent and significant limitation of this kind of statistical analysis is that the inferred equivalence classes become easily noisy as their support (total number of occurrences) or size (total number of nodes) decreases. Indeed, ExAlg's template inference process described is extremely brittle to the presence of noisy equivalence classes and cannot deal with singleton collections of input pages.

As in the original proposal, we use thresholds on the minimum size and on the minimum support to prevent the algorithm from generating too many noisy classes.[2] However, it is worth noticing that in our setting the template analysis is just an optimization aiming at reducing the number of generated extraction rules and that we have empirically observed that a single result page, usually containing up to tens of records, is already enough for the specific goals of our analysis.

The classes whose occurrence-vector exactly reports the number of objects published per page are called the *root* equivalence classes: their nodes occur exactly once in every detail page and as many times as the number of result records in every result page (an information available as part of RED's input). We classify as template nodes those occurring in the root equivalence classes or in any other equivalence class that appears less frequently than the root equivalent class in every page.[3]

Example 3.1. The root equivalence classes for the running example are ϵ_0^r and ϵ_0^d reported in Figure 4; the resulting template nodes are depicted with gray background in Figure 3. Notice that all their nodes are template nodes. $\epsilon_2^d, \epsilon_3^d, \epsilon_4^d, \epsilon_5^d$ are discarded because their support is too small and indeed they are noisy classes listing values that occur more than once by chance. ϵ_3^r is discarded because it occurs more frequently than the root equivalence class ϵ_0^r. Also nodes in ϵ_1^r, ϵ_2^r are correctly classified as template nodes. Notice that nodes in ϵ_4^r and ϵ_5^r are erroneously classified as template nodes.

The wrong classification of a value as template node will not prevent the generation of correct rules for the corresponding attribute as long as other occurrences of the values of that attribute are correctly classified as target values.

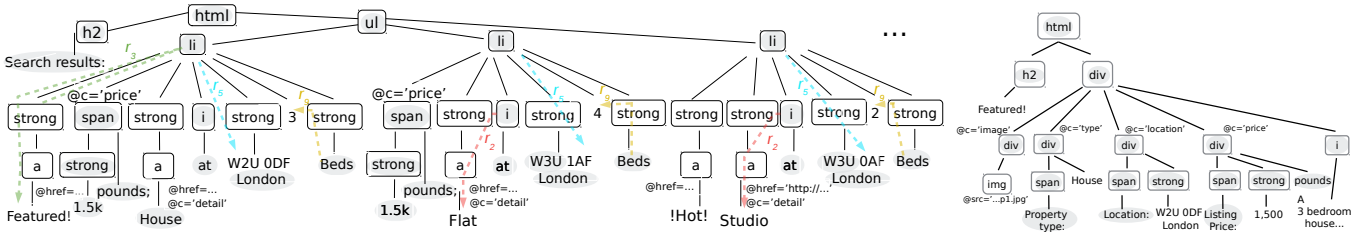
3.2 Extraction Rules Enumeration

The rules generation algorithm builds on the output of the template analysis to enumerate result (respectively, detail) extraction rules associated with tree-paths starting from a uniquely identifiable template node within the result records (resp., detail page), named *pivot*, and reaching a non-template *target value* node.

[1] Text nodes are split in several contiguous siblings nodes by tokenizing at word level.

[2] We consider only equivalence classes whose support is at least 20% of the input objects, but never fewer than 2, and only classes including at least 3 nodes.

[3] These classes are those associated with the *main* data structure of the two types of pages, i.e., the list of records of the result pages and the flat record of the detail pages, or those optional attributes nested within them.



(a) A portion of the DOM tree of the result page.

(b) DOM of the detail page 1.

Figure 3: Running example: DOM of input pages.

class	*nodes*	*occ. vector*
ϵ_0^r	{LI, I, at, A[@class = "detail"], Beds}	[6]
ϵ_1^r	{HTML, UL, H2, Search, results :, . . .}	[1]
ϵ_2^r	{SPAN[@class = "price"], STRONG, pounds}	[4]
ϵ_3^r	{STRONG}	[12]
ϵ_4^r	{House, A, 1.5k, W3U, E15, 2XD}	[2]
ϵ_5^r	{Oxford, London}	[3]
ϵ_0^d	{Property, type :, Location :, . . .}	[1, 1, 1, 1, 1, 1]
ϵ_1^d	{DIV[@class = "price"], SPAN, STRONG, Listing, Price :, pounds}	[1, 1, 0, 1, 1, 0]
ϵ_2^d	{House}	[1, 0, 0, 1, 0, 0]
ϵ_3^d	{1, 500}	[1, 1, 0, 0, 0, 0]
ϵ_4^d	{London}	[1, 1, 1, 0, 0, 0]
ϵ_5^d	{Oxford}	[0, 0, 0, 1, 1, 1]

Figure 4: Running example: template analysis.

r_0 : //span[contains(.,"pounds")]/../*[1]/text()[1]
r_1 : //li/*[2]/*[1]/text()[1]
r_2 : //i/ps::*[1]/*[1]/text()[1]
r_3 : //span[@class="price"]/../*[1]/*[1]/text()[1]
r_4 : //span[@class="price"]/*[1]/text()[1]
r_5 : //li/*[4]/text()[1]
r_6 : //i[contains(.,"at")]/fs::*[1]/text()[1]
r_7 : //i[contains(.,"at")]/*[4]/text()[1]
r_8 : //span[@class="price"]/fs::*[3]/text()[1]
r_9 : //strong[contains(., "Beds")]/ps::text()[1]
l : //a[@class="detail"]

(a) Result candidate rules.

d_0 : //span[contains(.,"Price:")]/../*[2]/text()[1]
d_1 : //span[contains(.,"type:")]/../text()[1]
d_2 : //h2/text()[1]
d_3 : //span[contains(.,"Location:")]/../fs::*[1]/text()[1]
d_4 : //span[contains(.,"Location:")]/fs::*[1]/text()[1]
d_5 : //span[contains(.,"Price:")]/../ps::*[1]/text()[1]
d_6 : //i/text()[1]
d_7 : //span[contains(.,"Location:")]/fs::*[2]/text()[1]

(b) Detail candidate rules.

Figure 5: Running example: extraction rules.

We use as candidate pivot every template node that is either a text, or an element with 'id' or 'class' attribute; every non-template text node is considered as a candidate target value. We then enumerate all the possible tree-paths leading from any pivot node to any target node by hopping over the tree along with a set of predefined XPath step expressions. Namely, we consider expressions capable of moving to the parent element node, to one of the children (either text or element), to the next/previous sibling element.

A couple of additional constraints on the tree-paths are enforced in order to reduce the number of enumerated tree-paths and discard those associated with incorrect rules. First, the path length is bounded by a threshold δ. Second, the analysis considers only a small subset of the input pages which is assumed unbiased.[4]

Each tree-path is translated into an executable extraction rule by appending several XPath step sub-expressions, eventually.

The first XPath step has to match the pivot, and takes one of the following forms depending on its node type: //e[contains(.,"v")] for a textual pivot v having e as parent node; //e[@id="v"] (resp., //e[@class="v"]), for an element pivot e having an id (resp., class) attribute valued v; just //e for any other element pivot.

Then it follows a sequence of XPath step expressions each after one of the hops composing the tree-path from the pivot to the target value: parent, child, following-sibling (abbreviated fs in Figure 5a), and preceding-sibling (ps); each step (except those on the parent axis) is also followed by an XPath positional predicate, e.g., '[4]', and by a node-test: The last step uses text() to target textual values, where all the other steps use '*' to select element nodes.

Example 3.2. Consider rule r_6 shown in Figure 5a: it is based on the 3 steps tree-path 'at ⇝ i ⇝ strong ⇝ W2U 0DF London' from the pivot node 'at' to the Location value. The XPath expression is obtained by combining: the expression selecting the pivot node (//i[contains(.,'at')]); the expression fs::*[1] to move to the next sibling element; and the final expression text()[1] to reach the target text value. A few tree-paths generating the result rules are shown directly in Figure 3 by means of colored dashed lines annotated with the rule and connecting the pivot to the target value across all the nodes in the path.

RED groups the generated rules by extracted values. Within a group producing the same values, only the rule associated with the shortest tree-path is retained, ties are broken by selecting the rule generated earlier. In the running example, r_7 is removed as it extracts the same values as r_5. The rationale to prefer shorter paths is that template nodes far away from the target values are

[4]We use at most 3 detail pages and 1 result page.

progressively less likely to generate rules that work reliably across all records and pages. More sophisticated criteria, such as those considering the robustness of the extraction rule [11, 22] could be adopted, but this is beyond the scope of the present paper.

4 REDUNDANCY SEEKING

RED analyzes the redundancy between all pairs of generated result and detail rules for identifying the likely correct rules. From an abstract point of view, a pair of result/detail rules can be considered *redundant* when the two rules end up extracting values associated with the same attribute of the abstract relation for every object.

Concretely, given one of such pairs composed of one result rule and one detail rule, a pairwise comparison of the extracted values is not trivial as they might extract a different number of values due to the presence of optional attributes. It is necessary to find out the correct alignment of the rules, i.e., where *nil* should be inserted to indicate a missing value within the list of values extracted by the result rule. We introduce a novel *soft segmentation* technique to find the alignment between result rules and detail rules, thus avoiding the tricky problem of finding the exact boundaries of every result record [20, 28]; then, we describe a score function for measuring the redundancy of a pair composed by one aligned result rule and one detail rule.

4.1 Soft Segmentation

The presence of optional attributes in the result pages can lead to the generation of rules that extract fewer values than the number of records. Soft segmentation is RED's technique for finding the correct alignment of the extracted values (w.r.t. the result records).

A prerequisite of the soft segmentation technique is the availability of exactly one detail link per each result record, i.e., that pointing to the corresponding detail page. RED leverages the knowledge of the detail pages, whose urls are assumed given as part of the input, to locate the detail link occurrences within the result records. As all occurrences must be associated with same "slot" of the underlying template, we have to disambiguate all the cases in which several copies of a detail link occur within the same result record, or optional detail links are present (e.g., for example in the first and third featured result records in Figure 1).

We apply the rules generation algorithm described in Section 3 by taking any of the link occurrences as target and generate a set of *link rules*. Only the rules extracting same number of links as the number of result records are considered, and in presence of several candidates, only the rule generated earliest is saved as link rule.

The soft segmentation technique finds out the correct alignment of the values extracted by a result rule w.r.t. detail the link occurrences during a traversal of the DOM tree. If a result rule is correct and extracts exactly the same number of values as the number of result records, the extracted values perfectly interleave with the detail link occurrences, i.e., either the values always occur before the links or they always occur after them. Conversely, in presence of result rules extracting fewer values, a correct alignment is needed.

Example 4.1. The link rule l of our running example is shown in Figure 5a. Let l_i denote the detail link node to the i-th detail page. Consider the correct rule r_9 extracting 6 values, in Figure 2b. It turns out that during an in-order traversal of the DOM tree in Figure 3a, detail links and extracted values interleave perfectly and occur as follows: $\langle l_1, 3, l_2, 4, l_3, 2, l_4, 3, l_5, 1, l_6, 2\rangle$. Instead, detail links and values extracted by r_8 do not: $\langle l_1$, W2U 0DF…, l_2, W3U 1AF…, $\underline{l_3}$, $\underline{l_4}$, E15 2CD…, l_5, E15 2XD…, $l_6\rangle$.

For dealing with these cases, *nil* values should be injected to reestablish the pattern that either every link precedes a value or vice-versa. At most two interleaving sequences of the values, called *alignments*, of a result rule, are possible: Given the result rule r, we use r^a (resp., r^b) to denote the alignment of the rule in which all nodes extracted by r are considered occuring *after* (resp., *before*) the corresponding detail link in every result record.

Example 4.2. Let s_8^a and s_b^b denote two sequences of exactly 12 elements (6 values plus 6 detail links) corresponding the alignment of rule r_8 w.r.t. detail links extracted by l in Figure 5a. Either every value occurs before a link:
$s_8^b = \langle \underline{nil}, l_1$, W2U…, l_2, W3U…, l_3, $\underline{nil}$, l_4, E15 …, l_5, E15…, $l_6\rangle$; or after a link:
$s_8^a = \langle l_1$, W2U…, l_2, W3U…, l_3, $\underline{nil}$, l_4, E15…, l_5, E15…, l_6, $\underline{nil}\rangle$.
We annotate r_8 to specify the relative position of the extracted values w.r.t. detail links so that the *nil* values can be properly inserted. Let p_r denote the result page of the running example:
$r_8^b(p_r) = \langle nil$, W2U 0DF…, W3U 1AF…, nil, E15 2CD…, E15 2XD…$\rangle$
$r_8^a(p_r) = \langle$W2U 0DF…, W3U 1AF…, nil, E15 2CD…, E15 2XD…, $nil\rangle$.

RED tries to enforce an admissible alignment to every produced result rule. Those rules that cannot be aligned are discarded.

Example 4.3. Consider the sequence of values extracted by r_1 and detail links in our running example:
$\langle$1.5k, l_1, Flat, l_2, Studio, l_3, House, l_4, Ensuite, l_5, l_6, at$\rangle$.
The presence of two consecutive links (l_5 and l_6) prevent values and links from interleaving perfectly. Thereby r_1 is removed.

Table 1 shows the alignments for all the result rules in Figure 5a w.r.t. link rule l for our running example.

Table 1: Aligned result rules after soft segmentation.

r_0^b:	$\langle$1.5k, 1.5k, nil, 2.05k, 1.2k, $nil\rangle$
r_2:	$\langle$House, Flat, Studio, House, Ensuite, Single Room$\rangle$
r_3^b:	$\langle$Featured!, 1.5k, nil, 2.05k, 1.2k, $nil\rangle$
r_5^a:	$\langle$at, W3U 1AF…, W3U 0AF…, E15 2CD…, E15 2XD…, $nil\rangle$
r_5^b:	$\langle nil$, at, W3U 1AF…, W3U 0AF…, E15 2CD…, E15 2XD…$\rangle$
r_6:	$\langle$W2U 0DF…,W3U 1AF…,W3U 0AF…,E15 2CD…,E15 2XD…,E18 2XD…$\rangle$
r_8^b:	$\langle nil$, W2U 0DF…, W3U 1AF…, nil, E15 2CD…, E15 2XD…$\rangle$
r_8^a:	$\langle$W2U 0DF…, W3U 1AF…, nil, E15 2CD…, E15 2XD…, $nil\rangle$
r_9:	$\langle$3, 4, 2, 3, 1, 2$\rangle$

The soft segmentation algorithm takes as input a result rule r, the link rule l and a set of result pages. It visits every page by ordering links and values into a sequence of occurrences and checks whether it is possible to inject *nil* values into the sequence to make values (including *nil*) and links to interleave perfectly. If so, it outputs at most two admissible alignments of the result rules, namely r^a and r^b (we simply write r where the alignments coincide).

4.2 Redundancy Score

Given a pair (r^*, d) composed of one aligned result rule (with $* \in \{a, b\}$) and one detail rule, and a set of objects O, the redundancy score of the pair is defined as the average pairwise score between their values:

$$\text{red}(r^*, d) = \sum_{o \in O} \frac{\text{score}(r^*(o), d(o))}{|O|}$$

where $d(o)$ and $r^*(o)$ denote the string extracted for object o by d and r^*, respectively. We then define the following *redundancy score* as a distance function over pairs of values:

$$\text{score}(v_1, v_2) = \begin{cases} 0 & \text{, if } v_1 \text{ substring of } v_2 \text{ or vice versa} \\ JS'(v_1, v_2) & \text{, otherwise} \end{cases}$$

where $JS'(v_1, v_2)$ is a *Jensen-Shannon* string distance modified for handling *nil*. If both v_1 and v_2 are *nil*, it returns 0. If either is *nil*, but not the other, it returns 1. In all other cases, it's a standard string distance. We noticed that the change of formats for the values of an attribute within the same site are very rare in real sites as they also tend to confuse the end user, i.e., the Price attribute is displayed as `1.5k` on a the result record whereas it is presented as `1,500` on the corresponding detail page of the running example in Figure 1. Hence, we observed only negligibly different results by adopting any other popular string distance function [7]. In practice, dealing with the substring cases is more important than the choice of the string distance function.

Example 4.4. Table 2 reports all the pairs scoring less than 0.4 ordered by their redundancy score for our running example. Each pair is composed of an aligned result rule from Table 1, and a detail rule from Figure 5b. The column "*Attributes*" reports whether the rules of the pair are correct, noisy, partially correct, or mis-aligned.

Table 2: Pairs of rules scoring less than 0.4.

pair π	red(π)	*Attributes*
(r_2, d_1)	0	Type correct
(r_6, d_4)	0	Location correct
(r_2, d_6)	0	Type vs Description
(r_8^b, d_7)	0	mis-alig. Location vs partial City
(r_8^a, d_5)	0	partial Location vs partial Location
(r_9, d_6)	0	Beds vs Description
(r_5^a, d_4)	0.333	noisy Location vs Location
(r_0^b, d_0)	0.397	Price correct

5 NOISE REMOVAL

RED uses the redundancy score to filter pairs of rules that are somewhat, but not sufficiently, redundant. Given a redundancy score threshold ρ, we consider as *not redundant* all the pairs of rules having a redundancy score greater than ρ. Unfortunately, many of the remaining pairs can still contain noisy rules: On one hand, there could be an incorrect alignment of the values, due to the presence of too many similar values in result records, e.g., (r_8^b, d_7) in Table 2; on the other hand, even noisy rules happen to be incidentally similar sometimes, e.g., (r_8^a, d_5) and (r_5^a, d_4). This is especially true when the range of the possible values of the compared attributes is rather limited; for example (the number of) Rooms, Beds, and Baths in the real estate domain are all small positive integers. Generally speaking, the correct result rules cannot be trivially separated from the incorrect ones with any fixed value of the threshold.

RED processes the redundant pairs by ascending redundancy score to remove the noisy pairs. First, the *result rule validation* analyzes whether a result rule has been correctly aligned and therefore properly fits in the HTML template of the result pages. Then, the *noise redundancy removal* leverages the redundancy scores to select for each attribute only the best pairs of rules, i.e., those having the lowest redundancy score.

5.1 Result Rules Validation

The result rules validation technique is based on the availability of some correctly aligned result rules as output of the previous processing steps. Beside the link rule, which is assumed to be correct, several other result rules might have only one alignment after the soft segmentation process, for example r_0^b and r_2 in Figure 1. RED leverages these already aligned and therefore inherently validated rules to validate other result rules.

A result rule r is validated against an already correctly-aligned rule g that is assumed correct. We can thereby infer two possible traversal sequences of their nodes corresponding to two positional alignments, and namely: s^b – extracted nodes of r are before those of g; and s^a – extracted nodes of r are after those of g. Then, as described in Section 4.1, we get the real traversal sequence s by traversing the DOM tree. If s does not matches any of s^b and s^a, r is not a valid result rule. We thereby remove r and all the redundant pairs containing r.

Example 5.1. For validating r_8^b based on r_9, RED considers two possible alignments of the values extracted by r_8^b w.r.t. r_9:
⟨3, 4, W2U 0DF..., 2, W3U 1AF..., 3, 1, E15 2CD..., 2, E15 2XD...⟩
⟨3, W2U 0DF..., 4, W3U 1AF..., 2, 3, E15 2CD..., 1, E15 2XD..., 2⟩ .
Neither of these matches the actual sequence in which these values occur on the page: ⟨W2U 0DF..., 3, W3U 1AF..., 4, 2, E15 2CD..., 3, E15 2XD..., 1, 2⟩.
Therefore, r_8^a fails to validate w.r.t. r_9 and is dropped.

5.2 Removing Redundant Noise

Once all the result rules have been validated, the surviving pairs include correct pairs together with partially correct and noisy redundant pairs that are incidentally similar.

The key assumption to remove the noisy pairs in the first category is that correct pairs have a better redundancy score than pairs containing noisy rules. The assumption is exploited in the first step of the algorithm removeRedNoise shown in Listing 1 (lines 2-9) and dealing with already validated result rules.

The second step of the algorithm (lines 10-17) prefers pairs of rules extracting fewer *nil* values: at this stage, all the noisy pairs have already been removed so that we can directly remove the result rules extracting a larger number of *nil*s. Notice that with the exception of the initial ordering (line 11 vs line 3), the two steps are performed by two almost identical loops (lines 2-9 vs lines 10-17). However, they cannot be merged because the latter loop expects as

Listing 1: removeRedNoise

```
Input: A set of redundant pairs Π = {(r, d) : red(r, d) < ϵ}.
Output: A set of correct rule pairs.
1  begin
2      Π′ ← ∅ ;
3      order pairs in Π by increasing redundancy score;
       /* Remove pairs with overlapping result DOM nodes */
4      foreach (r, d) ∈ Π do
5          remove (r, d) from Π ;
6          foreach (r′, d′) ∈ Π do
7              if DOMoverlap(r, r′) then
8                  remove (r′, d′) from Π ;
9          add (r, d) to Π′ ;                    /* saves only the best pair. */
       /* Prefer pairs whose detail rule extracts less nils.  */
10     Π″ ← ∅ ;
11     order Π' by decreasing n. of nil in the detail rules;
12     foreach (r, d) ∈ Π′ do
13         remove (r, d) from Π′ ;
14         foreach (r′, d′) ∈ Π′ do
15             if DOMoverlap(r, r′) then
16                 remove (r′, d′) from Π′ ;
17         add (r, d) to Π″ ;
18     return Π″;
```

input a list of pairs from which the noisy rules have been already removed, as produced by the first loop.

After the two processing steps, the surving pairs are the output result rules that RED classifies as correct.

Example 5.2. Consider again two pairs involving d_4 in Example 4.4, i.e., (r_6, d_4) and (r_5, d_4). The first step of removeRedNoise (lines 2-9) removes the latter pair, i.e., the pair with the worst redundancy score, since r_5 and r_6 extract overlapping nodes and only one of them is assumed to be correct.

Both pairs (r_6, d_4) and (r_8^a, d_5) are 0-scored and r_6 and r_8^a are two overlapping result rules: at most one of them can be correct. Both pairs reach the second loop (lines 10-17). The (r_8^a, d_5) pair is then removed because it contains the largest number of *nil* elements.

Table 3: Pair of rules after RED's noise removal step.

pair π	red(π)	note
(r_2, d_1)	0	Type
(r_6, d_4)	0	Location
(r_2, d_6)	0	Type vs Description
(r_9, d_6)	0	Beds
(r_0^b, d_0)	0.36	Price

Example 5.3. Table 3 reports (ordered by the redundancy score) the pairs after all the steps of the noise removal process. The pairs (r_2, d_1) and (r_2, d_6) are both 0-scored and the correct rule r_2 for Type is included in the output for its redundancy with the Description, an attribute on the detail pages that often "includes" the Type as a substring. So RED output on the running example is the set of aligned result rules r_0^b, r_2, r_6 and r_9 together with the link rule l.

6 EVALUATION

We inspect RED's characteristics and performance through a series of experiments on real-world websites. We first evaluate RED on a dataset derived from the one used in DIADEM [18], a state-of-the-art ontology-based data extraction system. We then apply RED to another dataset comprised of websites from a variety of application domains, a challenging setting for ontology-based data extraction systems that require some supervision on the level of the application domain. We show that, in contrast, RED can attain consistently high precision and recall over many domains.

As for the comparison with other fully unsupervised approaches beside DIADEM, we also considered DEPTA [41], another unsupervised system that, to the best of our knowledge, deals with the most similar setting to ours: unsupervised extraction from result pages, even a single result page.

6.1 Evaluation method

Datasets: We evaluate RED on two datasets, named DIA_DS and RED_DS, including 130 websites in total. DIA_DS consists of 100, still reachable websites from the original DIADEM's dataset, half in the REAL_ESTATE, half in the USED_CARS domain. We use DIADEM to generate wrappers on these sites and collect the detail pages for the dataset, as needed by RED.

RED_DS consists of 30 sites from 10 domains, obtained by randomly picking 3 sites from the *Alexa Top* 100 *Global Sites*. We exclude sites where the detail pages are not hosted on the site itself but rather refer to other sites (typically metasearch portals), as these cases do not fit RED's assumption that detail pages share a common template. Each test-case is related to a single website and it consists of one result page and the corresponding set of detail pages. All the records listed in the result pages are obtained by following the default sorting criteria from the website. Both datasets can be found on https://github.com/redwww/experiments.

Optional Attributes: To validate the need for dealing with optional attributes in result pages, we observed that in our datasets there is a considerable number of sites that contain optional attributes. In detail, 17% of sites in DIA_DS (i.e., 20 sites) and 36% of those in RED_DS (i.e., 13 sites) contain optional attributes. This well illustrates the importance of approaches, such as the soft segmentation technique, dealing with optional attributes in result pages.

Metrics: For each attribute of the result pages in our datasets, we manually crafted the correct XPath expression to extract correct data as golden standard, based on which we computed the number of true positives (tp), false negatives (fn), false positives (fp), precision $P = tp/(tp + fp)$, recall $R = tp/(tp + fn)$, and the F_1−measure, as $F = (2 \cdot P \cdot R)/(P + R)$.

These metrics have been computed both at (macro or) rule level and at (micro) level of extracted values. At rule-level, we consider a rule correct only if it extracts a set of values perfectly matching with that extracted by the golden rule, without any missing value: any rule extracting just a few noisy values or missing a single value, is considered the same as if it were completely wrong. We also calculate all the metrics at value-level by comparing the number of correct/noisy extracted values. In Table 4 and in Table 5 we report the main results for the two considered datasets: P, R, F stand for the precision, recall and F-measure, reported both at rule-level (e.g., F_r) and at value-level (e.g., F_v).

Table 4: RED vs DIADEM Performance.

System		DIADEM	RED	RED
Target Attributes		Ontology	Ontology	All
REAL_ESTATE	P_r	**0.94**	0.91	0.92
	R_r	**0.90**	**0.90**	0.91
	F_r	**0.92**	0.91	0.91
	P_v	**0.98**	**0.98**	0.98
	R_v	0.92	**0.94**	0.93
	F_v	0.95	**0.96**	0.96
USED_CARS	P_r	**0.97**	0.96	0.97
	R_r	**0.96**	**0.96**	0.96
	F_r	**0.96**	**0.96**	0.97
	P_v	**0.97**	0.96	0.97
	R_v	0.94	**0.98**	0.97
	F_v	0.95	**0.97**	0.97

6.2 Comparison with DIADEM

For a fair comparson between RED and DIADEM, we limit the target attributes to those in the ontology used by DIADEM, but also report RED's performance on all available attributes.

Table 4 presents $P/R/F$ results of RED and DIADEM systems over the dataset DIA_DS, both at the rule-level and at the value-level. Generally, RED and DIADEM perform roughly at the same high precision, recall, and F-measure. Considering that RED uses no domain ontology or other prior knowledge about the domain, that is quite remarkable. Moreover, RED can extract *all* the redundant attributes published in the result pages (as shown in the ALL column), not just the ones covered by the DIADEM ontology, achieving similar results. Overall, RED identifies 33% more attributes (about 800 attribute extraction rules) than DIADEM (about 600). The same close performance can be observed at site-level: RED extracts *all* attributes correctly in 73% of the websites, compared to 76% for DIADEM.

6.2.1 Failures & Limitations. The most frequent errors for RED and DIADEM systems can be grouped in two main categories.

Noise: DIADEM leverages its ontology to automatically annotate target pages. Wrong extracted values from DIADEM system are mainly due to misleading annotations. In our experiments, there are 5 sites where it suffers of this problem, e.g., top-lettings.co.uk. DIADEM extracts the template nodes `Available:` always occurring before the attribute AvailableDate and it wrongly takes it as a value of the attribute PropertyStatus. In contrast, RED's most common type of noise are partially correct rules, that extract only a fraction of the correct values.

Missing rules: Where DIADEM mostly misses attributes due to a lack of overlap with the background ontology, RED mostly misses attributes where the sites do not conform to its assumptions and specifically that result attributes also occur on the detail page and that attributes have a large enough domain to show some variability between result records. On site davidtompkins.co.uk the Description values in the result pages do not occur in detail pages. On cotswoldlettings.co.uk the Status of the properties is an optional attribute that can only take the value `SOLD` when present. RED wrongly considers all these values as part of the template.

6.3 Multi-domain evaluation

We further evaluate RED on RED_DS, a dataset covering 10 domains (Coffee, Concerts, Florist, Jewerlry, Job Search, Threatres, Books, Camera, Lighting, Sigars) with 3 randomly selected sites for each application domain. For space reasons, in Table 5 we just report the results for one example domain Cigars,[5] the quality metrics averaged over all the sites computed both at rule and value level, and other aggregated results; namely: total number of records (n) contained in the result page, total number of correct (tp), missing (fn) and incorrect (fp) values/rules produced.

Overall, RED attained a high performance at both rule-level (0.93 of F-measure) and value-level (0.96 of F-measure). At the value-level the precision is remarkable high (0.99).

Among 8 noisy rules in the output produced by RED, only 3 of them are entirely incorrect while all the other noisy rules are partially correct rules. On ncfjobs site, RED seems to perform badly as there are 4 noisy rules, out of which 3 are partially correct rules for Description, and 1 for JobTitle. All these noisy rules survived the noise-removal step because RED rules generation algorithm was not complete, i.e., it could not find correct rules for these attributes.

6.3.1 Comparison with other unsupervised approaches. For comparison with state-of-the-art unsupervised data extraction systems, we ran DEPTA [41] on the dataset RED_DS. This system is also capable of extracting records from result pages and does not require multiple sample pages. Although the algorithm was proposed in 2006, we compared with a runnable version re-implemented in 2012.[6] Since it blindly outputs multiple tables without telling the user which one contains the target data, we manually picked a single table containing the published records and evaluated precision, recall and F-measure at the *column*-level, to get scores comparable with RED's rule-level scores. This yields a negligible advantage for DEPTA as it is usually fairly easy to pick out the dominant table.

We evaluate DEPTA's record identification step over 14 out of 30 sites considered on which the prototype was able to produce a correctly-aligned table. Among these 14 sites, DEPTA achieves a respectable precision of 0.70 and recall of 0.93.

6.4 Robustness to parameter setting

RED depends on two key parameters: δ is the max allowed pivot-to-value distance used during the generation of the rules; ρ is the minimum threshold on the score to consider a pair of result/detail rules as not redundant during the search for correct pairs of rules.

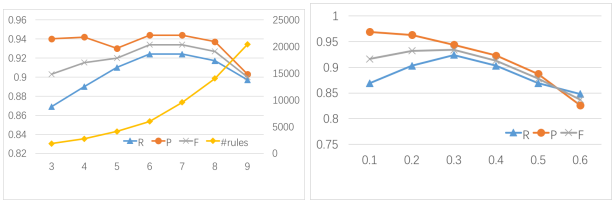
Figure 6a plots the performance of RED on the RED_DS dataset over several δ values (with fixed $\rho = 0.3$). While the precision is only slightly affected, the recall is significantly related to the maximum pivot-to-value distance. The larger the number of generated rules, the higher is the probability of generating correct rules. Unfortunately, it turns out that by generating too many rules, RED may end up introducing so many noisy rules that the probability of a noisy rule to be erroneously considered correct is increased, causing a loss in precision. In our experiments, $\delta = 6$ proved to be the sweet spot.

The other key parameter is the maximum redundancy score, ρ. A pair of result/detail rules is considered not redundant if its redundancy score is above ρ. It significantly affects the noise removal step whose task becomes harder and harder as noisy pairs are injected in the RED's processing pipeline. As shown in Figure 6b (with fixed

[5] A complete table can be found at https://github.com/redwww/experiments
[6] http://seagatesoft.blogspot.it/2012/05/structured-data-extractor.html

Table 5: Results of RED on RED_DS.

Domain	*Site*	*n*	tp_r	fn_r	fp_r	P_r	R_r	F_r	tp_v	fn_v	fp_v	P_v	R_v	F_v
Cigars	famous-smoke	60	8	1	1	0.88	0.88	0.88	439	15	20	0.95	0.97	0.96
	cigars-of-cuba	9	9	0	1	0.90	1	0.95	71	0	6	0.92	1	0.96
	vipcigars	18	3	0	0	1	1	1	54	0	0	1	1	1
10 domains	*30 sites*	*724*	*145*	*11*	*8*	*0.94*	*0.92*	*0.93*	*3365*	*248*	*28*	*0.99*	*0.93*	*0.96*



(a) Max pivot-value distance δ **(b) Max redundancy score ρ**

Figure 6: Performance by varying the two key parameters.

$\delta = 6$) the precision starts with maximum values, and then it gradually decreases as ρ becomes larger and larger. The noise removal process can still manage to remove most of the noises when ρ is at 0.4 but for larger values of the parameter there is a loss of precision, as the greater number of noisy pairs makes that process less effective. Conversely, too small values for ρ, meaning a too strict criteria for considering a pair of rules as redundant, makes RED wrongly remove some correct but not perfectly redundant pairs, as is evidenced by the recall loss of the plot in Figure 6b at $\rho < 0.3$.

For RED experimental evaluation, we measured RED performance by using the best empirically known values for the two parameters δ and ρ, i.e., $\rho = 0.3$ and $\delta = 6$.

Efficiency: By considering the number of input pages as a constant, RED efficiency can be conveniently analyzed in term of the initial number of rules generated. On RED_DS dataset, it generates an average number of 1259.17 result rules and 4912.10 detail rules, which reduce to 205.50 and 289.03 distinct vectors, respectively, after duplicates removal step; only 88.87 candidate pairs have a redundancy score smaller than ρ. RED achieves an average running time of 72.6 secs on a 64-bit Ubuntu system with an 8-core Intel *i*7 CPU at 3.40GHz, processing most sites in less than 30 seconds. RED running time is spent mostly on executing the generated extraction rules, a step that could be easily parallelised.

7 RELATED WORK

Web data extraction approaches and tools has been proposed both from research community and from industry [15, 16]. Due to the space limitation, we compare RED with the most related work and systems such as DEPTA [41], and DIADEM [18], which tackle with a similar problem as ours.

DEPTA executes two steps on a single result page, (*i*) identifying data records, and (*ii*) aligning values belonging to the same attribute. The first step, which is an enhancement of the MDR algorithm [29], works on a tag tree based on the visual containment relations of different HTML elements. It exploits the similarity of tag strings of nodes (and their descendant nodes) to find the data records. A further partial tree alignment method based on tree-edit distance is applied for the second step. DEPTA has two major drawbacks: First, it is highly sensitive even to small exceptions in the structure of the template (e.g., the record of some featured products may be presented differently from others) and optional nodes (e.g., the discount) in the repeated patterns. Second, due to the lack of any cues for identifying target values, it cannot easily separate target values from template nodes and other noises in the input pages.

One of the state-of-the-art system for domain-specific data extraction from result pages is DIADEM. We limit our discussion to its component most related to RED, i.e., that extracting data from result pages. DIADEM adopts automatic annotators to improve the data identification process, and requires the writing of a domain-dependent ontology which has to be designed and maintained by an expert. We exploit the intra-site highly precise redundancy to make RED more accurately target the relevant data without using any domain dependent feature.

Unsupervised approaches like RoadRunner [8], ExAlg [2], FiVaTech [27], TRINITY [35], and ViDE [30] all are based on the observation that pages providing homogeneous data are generated by using regular HTML templates, so they end up being mostly similar. It turns out that by analyzing the differences among a bunch of pages sharing a common template, it is then possible to reverse-engineer a model of that template, for example by means of regular expressions [2, 9]. However, in practice all these systems are not used for large scale extraction tasks because their performances are highly dependent on how well the input pages satisfy the underlying assumptions, to the point that the output quality level is unpredictable at scale [10]. In contrast, RED does not infer a precise description of the template, and it does not rely on it to find the data as all these latter systems, including ExAlg, do.

The work in [11, 12, 22] addresses the problem of XPath rules generation problem, but starts from a different input, i.e., a set of example nodes most of which are likely correct target nodes. These are usually provided by automatic annotators which are not available for every domain.

The approaches in [5, 23] and in [3] exploit content and/or schema redundancy and focus on template-based sites as RED does. However, they solve a completely different set of problems by adopting techniques based on the availability of data about the same objects across *several* autonomous sites. In other words, they focus on *inter*-site redundancy rather than on *intra*-site redundancy. It is therefore crucially important to align the different formats (e.g., Height expressed in inches on a site vs the same attribute expressed in centimeters on another site), a problem that we show is negligible intra-site. Also, since both approaches work directly on detail pages, they do not address record segmentation problems at all.

In [28] it is tackled the problem of segmenting result pages into records by exploiting links to detail pages exactly as RED. However, differently from RED, the authors do not consider at all the problem of extracting data from the result records once segmented; rather they describe two different approaches for tackling the segmentation problem: one formulates the task as a constraint satisfaction problem (CSP) and the other uses a probabilistic inference setting.

REFERENCES

[1] Manuel Álvarez, Juan Raposo, Alberto Pan, Fidel Cacheda, Fernando Bellas, and Víctor Carneiro. 2007. Crawling the content hidden behind web forms. In *Proc. of ICCSA*. 322–333.

[2] Arvind Arasu and Hector Garcia-Molina. 2003. Extracting Structured Data from Web Pages. In *SIGMOD Conference*. 337–348.

[3] Mirko Bronzi, Valter Crescenzi, Paolo Merialdo, and Paolo Papotti. 2013. Extraction and Integration of Partially Overlapping Web Sources. *PVLDB* 6, 10 (2013), 805–816. http://www.vldb.org/pvldb/vol6/p805-bronzi.pdf

[4] Andrea Calì and Davide Martinenghi. 2010. Querying the deep web. In *Proc. of EDBT*. 724–727.

[5] Shui-Lung Chuang, Kevin Chen-Chuan Chang, and ChengXiang Zhai. 2007. Collaborative Wrapping: A Turbo Framework for Web Data Extraction. In *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, Rada Chirkova, Asuman Dogac, M. Tamer Özsu, and Timos K. Sellis (Eds.). IEEE Computer Society, 1261–1262. https://doi.org/10.1109/ICDE.2007.368988

[6] Shui-Lung Chuang, Kevin Chen-Chuan Chang, and ChengXiang Zhai. 2007. Context-aware Wrapping: Synchronized Data Extraction. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB '07)*. VLDB Endowment, 699–710. http://dl.acm.org/citation.cfm?id=1325851.1325931

[7] William W. Cohen, Pradeep Ravikumar, and Stephen E. Fienberg. 2003. A Comparison of String Distance Metrics for Name-Matching Tasks. In *Proceedings of IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03), August 9-10, 2003, Acapulco, Mexico*, Subbarao Kambhampati and Craig A. Knoblock (Eds.). 73–78. http://www.isi.edu/info-agents/workshops/ijcai03/papers/Cohen-p.pdf

[8] Valter Crescenzi and Giansalvatore Mecca. 2004. Automatic Information Extraction from Large Websites. *J. ACM* 51, 5 (2004), 731–779.

[9] Valter Crescenzi and Paolo Merialdo. 2008. Wrapper Inference for Ambiguous Web Pages. *Applied Artificial Intelligence* 22, 1&2 (2008), 21–52.

[10] Valter Crescenzi, Paolo Merialdo, and Disheng Qiu. 2015. Crowdsourcing large scale wrapper inference. *Distributed and Parallel Databases* 33, 1 (2015), 95–122. https://doi.org/10.1007/s10619-014-7163-9

[11] Nilesh N. Dalvi, Philip Bohannon, and Fei Sha. 2009. Robust web extraction: an approach based on a probabilistic tree-edit model. In *Proc. of SIGMOD*. 335–348.

[12] Nilesh N. Dalvi, Ravi Kumar, and Mohamed A. Soliman. 2011. Automatic Wrappers for Large Scale Web Extraction. *PVLDB* 4, 4 (2011), 219–230.

[13] Steven DeRose and James Clark. 1999. *XML Path Language (XPath) Version 1.0*. W3C Recommendation. W3C. http://www.w3.org/TR/1999/REC-xpath-19991116.

[14] Xin Luna Dong and Divesh Srivastava. 2013. Big Data Integration. *Proc. VLDB Endow.* 6, 11 (Aug. 2013), 1188–1189. https://doi.org/10.14778/2536222.2536253

[15] Ruslan R. Fayzrakhmanov, Emanuel Sallinger, Ben Spencer, Tim Furche, and Georg Gottlob. 2018. Browserless Web Data Extraction: Challenges and Opportunities. In *Proceedings of the 2018 World Wide Web Conference (WWW '18)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 1095–1104. https://doi.org/10.1145/3178876.3186008

[16] Emilio Ferrara, Pasquale De Meo, Giacomo Fiumara, and Robert Baumgartner. 2014. Web data extraction, applications and techniques: A survey. *Knowl.-Based Syst.* 70 (2014), 301–323. https://doi.org/10.1016/j.knosys.2014.07.007

[17] Tim Furche, Georg Gottlob, Giovanni Grasso, Xiaonan Guo, Giorgio Orsi, and Christian Schallhart. 2012. OPAL: automated form understanding for the deep web. In *Proceedings of the 21st international conference on World Wide Web (WWW '12)*. ACM, New York, NY, USA, 829–838. https://doi.org/10.1145/2187836.2187948

[18] Tim Furche, Georg Gottlob, Giovanni Grasso, Xiaonan Guo, Giorgio Orsi, Christian Schallhart, and Cheng Wang. 2014. DIADEM: Thousands of Websites to a Single Database. *PVLDB* 7, 14 (2014), 1845–1856. http://www.vldb.org/pvldb/vol7/p1845-furche.pdf

[19] Tim Furche, Georg Gottlob, Giovanni Grasso, Giorgio Orsi, Christian Schallhart, and Cheng Wang. 2011. Little Knowledge Rules The Web: Domain-Centric Result Page Extraction. In *Proc. of RR*. 61–76.

[20] Tim Furche, Georg Gottlob, Giovanni Grasso, Giorgio Orsi, Christian Schallhart, and Cheng Wang. 2012. AMBER: Automatic Supervision for Multi-Attribute Extraction. *CoRR* abs/1210.5984 (2012).

[21] Tim Furche, Giovanni Grasso, Andrey Kravchenko, and Christian Schallhart. 2012. Turn the page: automated traversal of paginated websites. In *International Conference on Web Engineering*. Springer, 332–346.

[22] 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, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, Fatma Özcan, Georgia Koutrika, and Sam Madden (Eds.). ACM, 773–784. https://doi.org/10.1145/2882903.2915214

[23] Pankaj Gulhane, Rajeev Rastogi, Srinivasan H. Sengamedu, and Ashwin Tengli. 2010. Exploiting content redundancy for web information extraction. In *Proc. of WWW*. 1105–1106.

[24] Bin He, Mitesh Patel, Zhen Zhang, and Kevin Chen-Chuan Chang. 2007. Accessing the Deep Web. *Commun. ACM* 50, 5 (May 2007), 94–101. https://doi.org/10.1145/1230819.1241670

[25] Yeye He, Dong Xin, Venkatesh Ganti, Sriram Rajaraman, and Nirav Shah. 2013. Crawling Deep Web Entity Pages. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining (WSDM '13)*. ACM, New York, NY, USA, 355–364. https://doi.org/10.1145/2433396.2433442

[26] Arnaud Le Hors, Philippe Le Hégaret, et al. 2004. *Document Object Model (DOM) Level 3 Core Specification*. W3C Technical Reports. W3C. https://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/.

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

[28] Kristina Lerman, Lise Getoor, Steven Minton, and Craig A. Knoblock. 2004. Using the Structure of Web Sites for Automatic Segmentation of Tables. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, June 13-18, 2004*, Gerhard Weikum, Arnd Christian König, and Stefan Deßloch (Eds.). ACM, 119–130. https://doi.org/10.1145/1007568.1007584

[29] B. Liu, R. Grossman, and Y. Zhai. 2003. Mining Data Records in Web Pages. In *Proc. of SIGKDD*. 601–605.

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

[31] Jayant Madhavan, David Ko, Lucja Kot, Vignesh Ganapathy, Alex Rasmussen, and Alon Halevy. 2008. Google's Deep Web Crawl. In *Proc. of the VLDB Endowment (PVLDB)*. 1241–1252.

[32] Jayant Madhavan, David Ko, Łucja Kot, Vignesh Ganapathy, Alex Rasmussen, and Alon Halevy. 2008. Google's deep web crawl. *PVLDB* 1, 2 (2008), 1241–1252.

[33] Ali Mesbah, Arie van Deursen, and Stefan Lenselink. 2012. Crawling Ajax-Based Web Applications through Dynamic Analysis of User Interface State Changes. *ACM Trans. Web* 6, 1, Article 3 (March 2012), 30 pages.

[34] Stefano Ortona, Giorgio Orsi, Marcello Buoncristiano, and Tim Furche. 2015. WADaR: Joint Wrapper and Data Repair. *PVLDB* 8, 12 (2015), 1996–2007. http://www.vldb.org/pvldb/vol8/p1996-ortona.pdf

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

[36] Karane Vieira, Luciano Barbosa, Altigran Soares da Silva, Juliana Freire, and Edleno Silva de Moura. 2016. Finding seeds to bootstrap focused crawlers. *World Wide Web* 19, 3 (2016), 449–474. https://doi.org/10.1007/s11280-015-0331-7

[37] Jiying Wang and Fred H. Lochovsky. 2003. Data extraction and label assignment for web databases. In *Proc. of WWW*. 187–196.

[38] Jiying Wang, Ji-Rong Wen, Fred Lochovsky, and Wei-Ying Ma. 2004. Instance-based Schema Matching for Web Databases by Domain-specific Query Probing. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30 (VLDB '04)*. VLDB Endowment, 408–419. http://dl.acm.org/citation.cfm?id=1316689.1316726

[39] Lin Wang, Ammar Hawbani, and Xingfu Wang. 2015. Focused Deep Web Entrance Crawling by Form Feature Classification. In *Big Data Computing and Communications*, Yu Wang, Hui Xiong, Shlomo Argamon, XiangYang Li, and JianZhong Li (Eds.). Springer International Publishing, Cham, 79–87.

[40] Ying Wang, Wanli Zuo, Tao Peng, and Fengling He. 2008. Domain-Specific Deep Web Sources Discovery. In *Fourth International Conference on Natural Computation, ICNC 2008, Jinan, Shandong, China, 18-20 October 2008, Volume 5*, Maozu Guo, Liang Zhao, and Lipo Wang (Eds.). IEEE Computer Society, 202–206. https://doi.org/10.1109/ICNC.2008.350

[41] Yanhong Zhai and Bing Liu. 2006. Structured Data Extraction from the Web Based on Partial Tree Alignment. *TKDE* 18, 12 (2006), 1614–1628.

[42] Zhen Zhang, Bin He, and Kevin Chen-Chuan Chang. 2004. Understanding Web Query Interfaces: Best-effort Parsing with Hidden Syntax. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data (SIGMOD '04)*. ACM, New York, NY, USA, 107–118. https://doi.org/10.1145/1007568.1007583

[43] Feng Zhao, Jingyu Zhou, Chang Nie, Heqing Huang, and Hai Jin. 2016. SmartCrawler: A Two-Stage Crawler for Efficiently Harvesting Deep-Web Interfaces. *IEEE Transactions on Services Computing* 9 (2016), 608–620.