Joint Repairs for Web Wrappers

Stefano Ortona*, Giorgio Orsi†, Tim Furche* and Marcello Buoncristiano‡

* *Department of Computer Science, Oxford University, United Kingdom*
firstname.lastname@cs.ox.ac.uk

† *School of Computer Science, University of Birmingham, United Kingdom*
g.orsi@bham.ac.uk

‡ *Dipartimento di Matematica, Informatica ed Economia, Universita della Basilicata, Italy*
marcello.buoncristiano@unibas.it

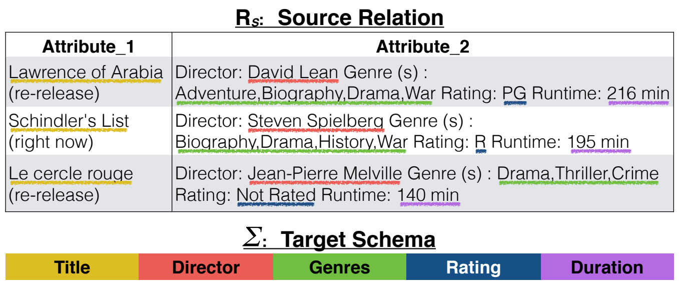
***Abstract*—Automated web scraping is a popular means for acquiring data from the web. Scrapers (or wrappers) are derived from either manually or automatically annotated examples, often resulting in under/over segmented data, together with missing or spurious content. Automatic repair and maintenance of the extracted data is thus a necessary complement to automatic wrapper generation. Moreover, the extracted data is often the result of a long-term data acquisition effort and thus jointly repairing wrappers together with the generated data reduces future needs for data cleaning. We study the problem of computing joint repairs for XPath-based wrappers and their extracted data. We show that the problem is NP-complete in general but becomes tractable under a few natural assumptions. Even tractable solutions to the problem are still impractical on very large datasets, but we propose an optimal approximation that proves effective across a wide variety of domains and sources. Our approach relies on encoded domain knowledge, but require no per-source supervision. An evaluation spanning more than 100k web pages from 100 different sites of a wide variety of application domains, shows that joint repairs are able to increase the quality of wrappers between 15% and 60% independently of the wrapper generation system, eliminating all errors in more than 50% of the cases.**

I. Introduction

Data acquisition plays an important role in modern organisations and is a strategic business process for data-driven companies such as insurers, retailers, and search engines. Data acquisition processes range from manual data collection and purchase, to cheaper but often technically challenging methods such as automated collection and crowdsourcing.

The abundance of web data has made *web scraping* (also known as web data extraction or web wrapping) an essential tool in data acquisition processes. A wrapper is a program that turns web content into structured data using techniques ranging from visual analysis of the rendered page to DOM tree mining [11], [18], [20], [35]. Web scraping is often the only viable data collection method for websites, in particular when no API is available. Although web scraping typically relies on inducing a wrapper for every source, a number of semi- or fully automated techniques for web scraping have emerged. These recent advances have finally allowed for accurate and fully automated wrapper induction at the scale of hundreds of thousands of sources [18]. They have also contributed to revitalised the area, as evident from a growing number of web scraping startups, e.g., Import.io, DiffBot, ScrapingHub.

This research has received funding from the EPSRC Programme Grant EP/M025268/1 - VADA, and the ERC PoC Grant N 641222 - ExtraLytics.



R_S: Source Relation

Attribute_1	Attribute_2
Lawrence of Arabia (re-release)	Director: David Lean Genre (s) : Adventure,Biography,Drama,War Rating: PG Runtime: 216 min
Schindler's List (right now)	Director: Steven Spielberg Genre (s) : Biography,Drama,History,War Rating: R Runtime: 195 min
Le cercle rouge (re-release)	Director: Jean-Pierre Melville Genre (s) : Drama,Thriller,Crime Rating: Not Rated Runtime: 140 min

Σ: Target Schema

Title	Director	Genres	Rating	Duration

Fig. 1: Web data extraction with RoadRunner

Automatically induced wrappers, however, are still more error prone than those created by humans, resulting in dirty data. As an example, Figure 1 (top) shows the outcome of the application of a wrapper generated by RoadRunner [11] on metacritic.com, a review aggregator. The extracted (source) relation (R_S) is given at the top, the desired (target) schema (Σ) at the bottom. Their differences exemplify common issues found in data extracted by automatically induced wrappers (see Section V for a more extensive analysis of the typical errors): **(1)** *Under/over segmented attributes* are commonly caused by irregularities in the HTML markup, or by the inability of wrapper induction systems to segment different attributes within the same DOM text node. **(2)** *Missing or incorrect column types* are associated with the lack of domain knowledge or suitable explicit labels in the source. **(3)** *Misplaced* and *missing* values (not present here) are typically caused by irregularities in the source, ambiguities in the content, or a narrow set of examples considered in the induction.

Repairing the data means aligning the source relation to the target schema—a task that requires the identification of exact values from R_S (underlined in Figure 1) and their placement under the correct attribute in Σ. However, just repairing the extracted data is often not sufficient, as in many cases data acquisition is an ongoing process and thus wrappers will likely be executed many more times in the future. This is particularly the case for data integration [3], [8], in which future executions of the repaired wrappers generate relations already aligned with the target schema, thereby reducing the need for repeated data cleaning and easing integration [15], [24].

The problem. This paper focuses on the problem of computing a *joint wrapper and data repair* for web data extraction systems (formally defined in Section III), so that future wrapper executions can benefit from the repairs applied to the wrapper. The problem takes as input a possibly faulty wrapper (specified in XPath), a relation generated by applying the wrapper to some collection of pages of the corresponding source, and a target relation schema. A solution to the problem is a set of repair expressions that align the relation to the target schema, adjusting the wrapper where possible.

Example 1. Consider the (source) relation of Figure 1, where we aim to align the underlined values to the target schema Σ. A possible repair is

$\langle \text{TITLE}, \text{substring-before}(\$, \texttt{␣()}) \rangle$
$\langle \text{DIRECTOR}, \text{substring-before}(\text{substring-after}(\$, \texttt{tor:␣}), \texttt{␣Gen}) \rangle$
$\langle \text{GENRES}, \text{substring-before}(\text{substring-after}(\$, \texttt{(s):␣}), \texttt{␣Rat}) \rangle$
$\langle \text{RATING}, \text{substring-before}(\text{substring-after}(\$, \texttt{ing:␣}), \texttt{␣Runt}) \rangle$
$\langle \text{DURATION}, \text{substring-after}(\$, \texttt{time:␣}) \rangle$

specifying, for each attribute: *(a)* the attribute label, and *(b)* an XPath string-valued expression that extracts the correct value from the concatenation ($) of the columns of Figure 1.

Novelty. Classical *wrapper maintenance* assumes that a correct (usually manually created) wrapper exists and focuses on maintaining the correctness of the wrapper through techniques such as *(a)* detecting changes on the sources that may affect the wrapper, and *(b)* minimally (and robustly) adjusting the wrapper to adapt to these changes [5], [22], [25], [27], [28]. Unfortunately, for automated wrapper induction methods, the induced wrappers are often incorrect. Tools for cleaning and maintaining the induced wrappers, as well as the extracted data, are primarily manual or supervised. Scalable tools for continuous maintenance of both the induced wrappers and the extracted data are still largely missing.

Contribution 1. We formally define the problem of computing maximal joint repairs and we give an optimal solution that relies on the existence of attribute oracles (Section II and III). Oracles are functions that can test whether a given value belongs to the domain of an attribute. We show that the problem is NP-complete but becomes tractable under a few natural assumptions based on the behaviour of wrapper induction systems: *(1)* errors in wrapper induced relations are likely to be systematic, as wrappers are often induced from templated websites; *(2)* wrapper induction systems tend to under-segment and misplace content rather than over-segment it (*atomic misplacements*, see Section V). In practice, this means that we can avoid reordering fields of generated relations in order to reconstruct a correct segmentation.

Despite being tractable, computing joint repairs remains difficult in practice. The main obstacle is the presence of attribute oracles, functions capable of recognising semantically related content in the relation, e.g., that in Figure 1 the tokens after "Runtime:" are all time durations. The recognition of semantically related content has some similarities to knowledge-based wrapper induction, where either an existing set of entity recognisers (like in [12], [18]) or cross-site, instance-level redundancy is employed to induce wrappers [3], [6], [8], [13]. However, in the wrapper induction setting, background knowledge is only compared with a small sample of webpages considered for induction. For repair, on the other hand, we can afford to compare it with all of the extracted data.

Contribution 2. We follow the inspiration of the former approaches and we propose an approximate solution that replaces oracles with an ensemble of entity recognisers [4], encoding existing background knowledge. This has advantages such as enabling per-source (and therefore per-wrapper) repairs, while retaining a broad applicability across domains.

Unfortunately, entity recognisers are often noisy and incomplete (as noted also in [4], [12], [18]) and our approach is specifically tailored to tolerate these deficiencies by eliminating noise and ambiguity and generalising the underlying structure to cope with incompleteness (see Section IV).

The resulting approach is fully unsupervised and computes an optimal approximation of the joint repairs using an ensemble of entity recognisers to *(1)* discover hidden attribute structures in the annotated relations through the elimination of incorrect annotations, and to *(2)* construct relation-wide repairs for both wrappers and relations.

Contribution 3. We show that our method is optimal under certain conditions on the error rate of the entity recognisers (Section IV) and, then, experimentally verify the performance and the generality of our method on a dataset of 100 websites from 10 different domains (Section V). The evaluation uses 4 different wrapper induction systems, namely DIADEM [18], DEPTA [35], ViNTs [38], and RoadRunner [11], and shows in each case an improvement in the quality of the extracted relation by 15%–60%, while fully repairing corresponding wrappers in more than 50% of the cases.

II. Definitions

Preliminaries. A *relation* R is a set of n-tuples $\langle u_1, \ldots, u_n \rangle$ where $n=\text{arity}(R)$ is the *arity* of the relation. A *schema* $\Sigma = \langle A_1, \ldots, A_m \rangle$ is a tuple representing *attributes* (or *fields*) of the relation, where $m=\text{arity}(\Sigma)$ is the arity of the schema. Each attribute $A \in \Sigma$ takes values from a *domain* $\text{dom}(A)$ extended with the null value. A domain is the set of admissible semantic values for a given attribute. As an example, the domain for the attribute MAKE of a car may contain the values "Ferrari", "Citroën" and "Ford". We test whether a value u belongs to a given domain via an *oracle* function ω_A, s.t., $\omega_A(u)=1$ if $u \in \text{dom}(A)$ or $u=\text{null}$, and $\omega_A(u)=0$ otherwise. We say that R *agrees* with Σ if $\text{arity}(R)=\text{arity}(\Sigma)$ and that R *satisfies* Σ if it agrees with Σ and $\omega_{A_i}(u_i)=1$, for each attribute $A_i \in \Sigma$ and corresponding attribute value u_i in any tuple $\langle u_1, \ldots, u_n \rangle \in R$.

Wrappers. In web data extraction, a wrapper is a program extracting structured data from a collection of (semi-structured) web pages P represented as DOM trees. Differently from, e.g., information extraction, in our setting the pages in P follow the same *template*. In this paper we limit ourselves to wrappers specified in XPath.

Definition 1. An XPath *wrapper* W is a set of pairs $\{\langle L_1, \chi_1 \rangle, \ldots, \langle L_m, \chi_m \rangle\}$, where each L_i is a label and each χ_i is an XPath expressions on DOM trees, possibly with functions, e.g., substring-after(//b[@name='price'],'£'). The application of a wrapper W to a collection of web pages P is denoted by W(P) and produces, as an output, a relation with schema $\langle L_1, \ldots, L_m \rangle$.

Fitness. Automatically induced wrappers often introduce misalignments between relations and schemas, i.e., *under/over segmented* or *misplaced* values. We therefore require a way to quantify the agreement between a relation and a schema.

Definition 2. The *fitness* for a tuple $\mathbf{u}$ (and for a relation R) w.r.t. a schema Σ and a set of oracles $\Omega = \{\omega_{A_1}, \dots, \omega_{A_{\mathsf{arity}(\Sigma)}}\}$ is computed as:

$$f(\mathbf{u},\Sigma,\Omega) = \sum_{i=1}^{c} \omega_{A_i}(u_i)/d \qquad f(R,\Sigma,\Omega) = \sum_{\mathbf{u}\in R} f(\mathbf{u},\Sigma,\Omega)/|R|$$

where $c = \min\{\mathsf{arity}(\Sigma), \mathsf{arity}(R)\}$ and $d = \max\{\mathsf{arity}(\Sigma), \mathsf{arity}(R)\}$.

Example 2. Consider the following (misaligned) relation and the schema $\Sigma : \langle \text{MAKE}, \text{MODEL}, \text{PRICE}\rangle$.

	MAKE	MODEL	PRICE
u_1		£19k	Audi A3 Sportback
u_2		£43k	Audi A6 Allroad quattro
u_3	Citroën	£10k	C3
u_4	Ford	£22k	C-max Titanium X

Given a set of oracles Ω=$\{\omega_{\text{MAKE}}, \omega_{\text{MODEL}}, \omega_{\text{PRICE}}\}$, the fitness of R w.r.t. Σ and Ω is 2/12 as only the two values of MAKE are in the correct position, out of 12 possible correct values. The other values are misplaced and/or under-segmented.

Repairs. The relation of Example 2 is misaligned and wrongly segmented w.r.t. to the schema but the correct information is somehow represented in the relation. When a relation does not fit a schema, we re-segment it and adjust the wrapper accordingly. Given a tuple $\mathbf{u} = \langle u_1, \dots, u_n\rangle \in R$, we denote by $\Pi(\mathbf{u})$ a tuple obtained by permuting the values of $\mathbf{u}$ and by $T(\mathbf{u})$ the string concatenation (on a suitable separator, e.g., ␣) $T(u_1)\|\dots\|T(u_n)$, where $T(u_i)$ is the string representation of $u_i \in \mathbf{u}$. Thus, $T(\Pi(\mathbf{u}))$ is the string computed by concatenating permuted values of $\mathbf{u}$ according to Π. A *segmentation* of a string S is a sequence $(s_1, \dots, s_k)$ of non-overlapping substrings of S. If the concatenation $s_1\|\dots\|s_k$=S, then we say that $(s_1, \dots, s_k)$ is a *partition* of S. We can now extend the notion of fitness to segmentations of strings: given a permutation Π of $\mathbf{u}$ and a segmentation of $T(\Pi(\mathbf{u}))$, the fitness of $T(\Pi(\mathbf{u}))$ can be computed by invoking oracles on elements of the segmentation rather than on elements of $\mathbf{u}$ (Definition 2).

Definition 3. Given a relation R and a schema Σ, a Σ-*repair* of R is a pair σ=$\langle\Pi,\rho\rangle$, where Π is a permutation of the fields of R and ρ a set of pairs $\langle A_i, \mathcal{E}_{A_i}\rangle$ where $A_i \in \Sigma$, and $\mathcal{E}_{A_i}$ is a regular expression over strings. The *application* of a Σ-repair σ on a tuple $\mathbf{u}$, denoted by $\sigma(\mathbf{u})$, is the tuple $\mathbf{u}^* = \langle \mathcal{E}_{A_1}(T(\Pi(\mathbf{u}))), \dots, \mathcal{E}_{A_m}(T(\Pi(\mathbf{u})))\rangle$, where $(\mathcal{E}_{A_1}(T(\Pi(\mathbf{u}))), \dots, \mathcal{E}_{A_m}(T(\Pi(\mathbf{u}))))$ is a segmentation of $T(\Pi(\mathbf{u}))$.

Put simply, a Σ-repair extracts values for an attribute A from a tuple $\mathbf{u}$. This is done by concatenating the (possibly permuted) values of $\mathbf{u}$, and by applying a regular expression $\mathcal{E}_A$. This notion generalises to relations, denoted by $\sigma(R)$, by applying σ to all tuples $\mathbf{u} \in R$.

Example 3. Consider the relation and the schema of Example 2. The following is a possible Σ-repair maximising the fitness of R w.r.t. Σ and Ω. The symbol \$ denotes the argument of the expression, and '␣' is the whitespace character:

$$\begin{array}{rl} \Pi: & (1,3,2) \\ \rho: & \langle \text{MAKE}, \mathsf{substring\text{-}before}(\$, ␣)\rangle \\ & \langle \text{MODEL}, \mathsf{substring\text{-}before}\ (\mathsf{substring\text{-}after}(\$, ␣), £)\rangle \\ & \langle \text{PRICE}, \mathsf{concat}(£, \mathsf{substring\text{-}after}(\$, £))\rangle \end{array}$$

The application of σ, to, e.g., u_4 produces the correct (segmentation and) tuple $\langle$Ford, C-max Titanium X, £22k$\rangle$.

Σ-repairs are applied to wrappers in a similar way.

Definition 4. The application of a Σ-repair σ to a wrapper $W = \{\langle A_1, \chi_1\rangle, \dots, \langle A_n, \chi_n\rangle\}$, denoted by $\sigma(W)$, produces a modified wrapper as follows:

$$\begin{array}{rl} W' = & \{\langle A_1, \mathcal{E}_1(\mathsf{concat}(\chi_{\pi_1}, \dots, \chi_{\pi_n})\rangle, \\ & \dots \\ & \langle A_m, \mathcal{E}_m(\mathsf{concat}(\chi_{\pi_1}, \dots, \chi_{\pi_n})\rangle\} \end{array}$$

where concat concatenates all the strings produced by the XPath expressions $\chi_{\pi_1}, \dots, \chi_{\pi_n}$ in the order specified by the permutation Π=$(\pi_1, \dots, \pi_n)$. In this work we limit ourselves to expressions that can be constructed by XPath 1.0 string manipulation functions, denoted by $f^{\Sigma}_{\text{XPATH}}$.

III. Maximal Joint Repairs

Definition 5. Let Σ be a schema, Ω a set of oracles for Σ, R a relation, and W a wrapper s.t. W(P)=R for a collection of web pages P. We say that a Σ-repair σ is a *maximal joint repair* for R and W w.r.t. Σ and Ω, if $\sigma(W(P))$=$\sigma(R)$ and σ maximises the fitness of R w.r.t. Σ and Ω, i.e., $f(\sigma(R), \Sigma, \Omega)$ is maximum.

As an example, the Σ-repair of Example 3 is a maximal joint repair as the fitness increases from 2/12 to (its maximum) 1.

We first establish the complexity of the problem when relations have only *atomic misplacements*, i.e., where an attribute value is either entirely misplaced or, if it has been over-segmented, the fragments are in adjacent fields in the relation (and in the correct order). Under this restriction, our problem is closely related to the *Table Induction Problem* (TIP) [6], [8], [13], [24], which asks to compute a segmentation of a list of strings in k parts in such a way that the values of each column minimise a given distance function.

TIP is shown to be NP-hard via a reduction from *Multiple Sequence Alignment* [6]. An optimal approximation can be obtained in polynomial time by decomposing the distance function in such a way that the independent minimisation of each component results in the minimisation of the whole distance function. In our setting, fitness maximisation plays the same role of distance minimisation, while the presence of the oracles reduces the fitness of a relation to an aggregate of the fitness of its tuples (i.e., we can independently maximise the fitness of individual tuples).

The problem is now to determine how hard it is to compute a maximal joint repair of a single tuple. We show that this problem can be reduced to the computation of a partition of $T(\mathbf{u})$ that, encoded as a Σ-repair, produces the maximal fitness. However, restricting to partitions raises the problem of treating "garbage" content—parts of the extracted data that are never accepted as value of an attribute in any segmentation and typically form part of the boilerplate code of a web page.

Example 4. Consider the following relation and the target schema $\Sigma : \text{R}(\text{BEDS}, \text{TYPE}, \text{CITY}, \text{PRICE})$.

	BEDS	TYPE	CITY	PRICE
u		2 beds apartment	Bradford	£495k

If the correct tuple is ⟨2, apartment, Bradford, £495k⟩, then it is not possible to obtain such a tuple by restricting to partitions of 4 elements of $\mathsf{T}(\mathbf{u})$ ="2␣beds␣apartment␣Bradford␣£495k", as the garbage substring "beds" must be part of an element of the partition (by definition of partition). We therefore need a way to identify and eliminate garbage substrings by leveraging the oracles. Since we have to enumerate all possible partitions, this can be done by marking substrings of $\mathsf{T}(\mathbf{u})$ that are never accepted by an oracle in any of the enumerated partitions. As an example, consider the following partitions of $\mathsf{T}(\mathbf{u})$:

	ω_{BEDS}	ω_{TYPE}	ω_{CITY}	ω_{PRICE}
Φ_1	"2 beds"	"apartment"	"Bradford £"	"495k"
Φ_2	"2"	"beds apartment"	"Bradford £"	"495k"
Φ_3	"2"	"beds"	"apartment Bradford"	"£495k"

If a substring is (part of) an attribute value, it will be accepted by an oracle in some of the enumerated partitions, e.g., "2" will be accepted by ω_{BEDS} in Φ_2 and Φ_3, while "beds" will never be accepted by any of the oracles. Once all partitions have been enumerated, we can update them by removing all identified garbage substrings. This will collapse some of the partitions as it is the case for Φ_1 and Φ_2 above, collapsing to ("2","apartment","Bradford £","495k").

Lemma 1: Let R be a relation with atomic misplacements, Σ a schema, k the arity of Σ, and Ω a set of oracles for Σ. Given a tuple $\mathbf{u} \in \text{R}$, we can construct a segmentation of $\mathsf{T}(\mathbf{u})$ of maximum fitness w.r.t. Σ and Ω in polynomial time.

Proof: We describe the polynomial time algorithm `Naïve-Repair` as a proof of Lemma 1. The algorithm first enumerates all possible partitions of $\mathsf{T}(\mathbf{u})$ of k elements. As we expect null values, we compute partitions with $k = \{1, \ldots, \mathsf{arity}(\Sigma)\}$ and fill the remaining elements of the partition (up to $\mathsf{arity}(\Sigma)$) with null. A partition of k elements for a string S can be computed recursively. Since each partition must contain at least one character, the 1-element partition consists of one element equal to S. A partition of k elements is computed by assigning to the first element of the partition the first y characters of the string ($y \in [1, \mathsf{length}(S) - k + 1]$), and computing the partitions of $k-1$ elements on the rest of the string. Once the partitions have been enumerated, `Naïve-Repair` tests each element of each partition against the oracles, and deletes those (garbage) substrings that are never accepted by an oracle in any of the partitions. Eventually the algorithm returns the partition (without garbage substrings) with maximum fitness. A full description of `Naïve-Repair` can be found online [1].

Under the assumption of atomic misplacements, the specific Π is not important because the correct value of an attribute will, in the worst case, be split across adjacent fields (we do not need to permute elements of $\mathbf{u}$). The number of different partitions is therefore equivalent to the number of ways a $\mathsf{T}(\mathbf{u})$ can be partitioned into $k = \mathsf{arity}(\Sigma)$ parts. Since the number of characters per tuple n is usually much larger than k, the number of different partitions is polynomial in n and described by the binomial coefficient $\binom{n+k-1}{k-1} \sim \frac{n^k}{\sqrt{k} \cdot k^k}$ [19]. ■

The interaction of attribute misplacements and over-segmentation makes computing an optimal solution intractable, since we have to find the right permutation of elements in $\mathbf{u}$ that, once segmented, produces the maximal fitness.

Example 5. Consider the following relation, having the same schema of Example 2, where values have been both wrongly segmented and misplaced.

	MAKE	MODEL	PRICE
u	Ford C-max Titanium	£22k	X

Without permuting the fields it is not possible to maximise the fitness, since the over-segmented and misplaced value "C-max Titanium X" cannot be tested against Ω_{MODEL}.

We can easily accommodate non atomic misplacements in Algorithm `Naïve-Repair` by enumerating all of the possible permutations of elements of $\mathbf{u}$ before partitioning $\mathsf{T}(\Pi(\mathbf{u}))$. This is unnecessary in table induction where the content is expected to be wrongly segmented but in the correct order. We now prove that the corresponding decision problem is NP-complete.

Theorem 1: Let R be a relation, Σ a schema, and Ω a set of oracles for Σ. Given a tuple $\mathbf{u} \in \text{R}$ and a score w, checking whether there exist a permutation $\Pi(\mathbf{u})$ and a segmentation of $\mathsf{T}(\Pi(\mathbf{u}))$ of fitness $\geq w$ w.r.t. Σ and Ω is *NP-complete*.

Proof: (Sketch) We prove the NP hardness via a reduction from the *Weighted Set Packing* problem (WSP). Given a set U and a family $\mathcal{S}$ of subsets of U, a packing is a subfamily $\mathcal{C} \subseteq \mathcal{S}$ of sets such that all sets in $\mathcal{C}$ are pairwise disjoint. If we assign a weight $w(S)$ to each element S of $\mathcal{S}$, the problem of determining if there exists a $\mathcal{C}_w$ of weight $\sum_{S \in \mathcal{C}_w} w(S)$ equal or greater than w is NP-complete. For any instance of WSP, we construct an instance of our problem as follows. Each element of U for WSP is taken as an element of $\mathbf{u}$. Each set S in $\mathcal{S}$ is mapped to a string x_S by considering the string representation of elements in U and concatenating them (in any order). We then define an oracle ω such that $\omega(x_S) = w(S)$ for each S in $\mathcal{S}$. Note that the fact that oracles return a positive real value does not affect the polynomial of the reduction. It can be shown that there exists a solution to WSP iff there exists a solution to our problem. The proof relies on the fact that pairwise disjoint sets are mapped to strings that do not share any element of $\mathbf{u}$, and such strings are therefore a segmentation of a permutation of $\mathbf{u}$. Membership in NP is straightforward. Given a set of substrings ϕ and a weight w, checking whether ϕ is non-crossing and that it has fitness w or more can be done in polynomial time. The complete proof of the Theorem can be found in [1]. ■

Once an optimal segmentation has been computed, we still have to construct the expressions matching the correctly segmented content in the relation. This can be done using the values of the segmentation as positive examples. Indeed an $f^{\Sigma}_{\text{XPATH}}$expression that exactly matches the correctly segmented values in each tuple can always be naïvely constructed by: *(i)* selecting the tuple with the content to be matched, and *(ii)* matching the desired value using substring functions that use the content before and/or after the correct value as context. The expression segmenting an attribute is then the disjunction of the expressions computed for each tuple.

Example 6. Consider again the relation R and the schema Σ of Example 2. The following is the correct segmentation of R obtained by executing `Naïve-Repair` on every tuple of R, and by mapping the elements of the segmentation to the corresponding attribute.

ω_{MAKE}	ω_{MODEL}	ω_{PRICE}
⟨Audi⟩	⟨A3 Sportback⟩	⟨£19k⟩
⟨Audi⟩	⟨A6 Allroad quattro⟩	⟨£43k⟩
⟨Citroën⟩	⟨C3⟩	⟨£10k⟩
⟨Ford⟩	⟨C-max Titanium X⟩	⟨£22k⟩

A naïvely constructed $f^{\Sigma}_{\text{XPATH}}$expression for, e.g., MAKE is:

matches($, £19k␣Audi␣A3␣Sportback) and
substring-after(substring-before($, A3␣Sportback), £19k␣)
or
...
matches($, Ford␣£22k␣C-max␣Titanium␣X) and
substring-before($, £22k␣C-max␣Titanium␣X))

Given the result of Lemma 1 and the fact that a Σ-repair for a relation with only atomic misplacements can always be constructed in polynomial time from the output of `Naïve-Repair`, we can now derive the following:

Theorem 2: Given a relation R with atomic misplacements, a wrapper W that generated R, a schema Σ and a set of oracles Ω for Σ, it is possible to construct a maximal joint repair for R and W w.r.t. Σ and Ω in polynomial time.

IV. Approximating Joint Repairs

Even under the assumption of atomic misplacements, computing a joint repair remains impractical. For instance, it is unrealistic to construct oracles for all attributes. Moreover, a tuple-by-tuple repair would still require testing $\mathcal{O}\left(\text{length}(\text{T}(\mathbf{u}))^{\text{arity}(\Sigma)}\right)$ alternative partitions against the oracles for each tuple $\mathbf{u}$, making it infeasible on large relations. E.g., in the book domain (Section V) we have roughly 2,000 records per site, 150 characters per record, and 5 attributes, i.e., $2,000 \cdot 150^5$ partitions.

Another problem is the robustness of the Σ-repair, i.e., its ability to match correct values on unforeseen data (or to avoid overfitting). Clearly, the naïve construction of expressions done by `Naïve-Repair` always overfits the examples and would require re-computing the repair at every wrapper execution as the produced expressions are never robust. Our goal is instead to produce a repair for the original wrapper such that future extractions on unforeseen data will produce clean values.

Approach Overview. We now introduce a general framework to *approximate* joint repairs that: *(i)* Relaxes the notion of oracles to allow errors. In particular, oracles are replaced by an ensemble of entity recognisers (annotators) tagging strings with *types* corresponding to attributes in Σ [4]. *(ii)* Under the assumption of atomic misplacements, computes robust relation-wide segmentations that can be encoded in $f^{\Sigma}_{\text{XPATH}}$.

Algorithm 1: Repair.

```
input : R – relation instance, set of tuples
input : Ω – ensemble of annotators
input : Σ – schema of the relation
input : $t_{flow}$ – flow network threshold
input : $t_{regex}$ – regex induction threshold
output: ρ – set of pairs $\langle A, \mathcal{E}_A \rangle$
1  $R_\Omega \leftarrow \emptyset$;  // Phase 1: Annotation
2  foreach $\mathbf{u} \in R$ do
3      anns($\mathbf{u}$) ← annotate($\mathbf{u}$, Ω);
4      $R_\Omega \leftarrow R_\Omega \cup \{(\mathbf{u}, \text{anns}(\mathbf{u}))\}$;
5  $R_{copy} \leftarrow R_\Omega$;  // Phase 2: Segmentation
6  $cover \leftarrow 0$; $S \leftarrow \emptyset$;
7  repeat
8      $\langle V, \lambda, E, w \rangle \leftarrow$ FlowNetwork($R_{copy}$);
9      s ← MaxFlowSequence($\langle V, \lambda, E, w \rangle$);
10     if $\nexists a, b \in s \mid a = b$ then
11         $S \leftarrow S \cup s$;
12     foreach anns ($\mathbf{u}$) $\in R_{copy}$ do
13         if subSequence(anns ($\mathbf{u}$), $s$) then
14             $cover \leftarrow cover + 1$;
15             $R_{copy} \leftarrow R_{copy} \setminus \{(\mathbf{u}, \text{anns}(\mathbf{u}))\}$;
16 until $cover \geq (|R| \cdot t_{flow})$ ;
17 foreach anns ($\mathbf{u}$) $\in R_\Omega$ do
18     foreach $\langle [i, j), A \rangle \in$ anns ($\mathbf{u}$) do
19         if $\nexists s \in S \mid A \in s$ then
20             anns($\mathbf{u}$) ← anns($\mathbf{u}$) $\setminus \{\langle [i, j), A \rangle\}$;
21 $\rho \leftarrow \emptyset$;  // Phase 3: Induction
22 foreach $A \in \Sigma$ do
23     $\rho \leftarrow \rho \cup \{\langle A, \text{regexInduction}(A, R_\Omega, t_{regex}) \rangle\}$;
24 return ρ
```

(iii) Optimally approximates (w.r.t. fitness) maximal joint repairs constructed by `Naïve-Repair`.

Relation-wide repairs require the understanding of the structure of the relation, e.g., to locate misplaced or wrongly segmented attribute values. In this respect, we observe the following: *(i)* Tuples are extracted from *templated* web pages, i.e., they follow a similar structure within a site modulo optional attributes, i.e., values may be omitted for some of the tuples. *(ii)* Wrappers make *systematic* errors, i.e., if an attribute is misplaced or wrongly segmented by a wrapper, this happens in a substantial fraction of the relation. *(iii)* The ensemble of annotators makes *non-systematic* errors.

Algorithm 1, given a relation R, a schema Σ and a set of annotators Ω, constructs an approximate joint repair under the assumption of atomic misplacements. The algorithm takes as input a relation R generated by the application of a wrapper W (not an input to the algorithm), and a collection Ω of annotators for values of attributes in Σ. Algorithm 1 works in three phases: *annotation*, *segmentation*, and *induction*.

Annotation. The input for this phase is a relation R and an ensemble of annotators Ω. For each tuple $\mathbf{u} \in R$, $T(\mathbf{u})$ is handed over to annotators in Ω producing, as an output, an annotated tuple (lines 1–4). We represent tuple annotations via a labelling function anns associating a tuple to a set of pairs of the form $\langle [i, j), A \rangle$, where $[i, j)$ is an integer interval (with $0 \leq i < j \leq \text{length}(T(\mathbf{u}))$) and $A \in \Sigma$. This phase produces an annotated relation R_Ω, consisting of a set of annotated tuples. We expect it to contain both incorrect and missing annotations.

Segmentation. The segmentation phase takes as input the annotated relation R_Ω. A straightforward way of constructing a Σ-repair of maximum fitness is to directly use the annotated values, since the fitness is determined by the number of times annotators recognise values in R_Ω. Due to noisy annotators, we first have to determine the sequences of annotation types representing the correct structure of the relation. The intuition is that correct annotations are very likely to be predominant and that errors, although frequent, are non-systematic

This can be encoded as the problem of computing max-flows in a network $N(V, E)$ having the following properties:

1) Each node $v \in V$ represents an annotation type $A \in \Sigma$ in a given context (i.e., within a sequence of annotations). Two special nodes, i.e., SRC (source) and SINK nodes represent the beginning and the end of the relation respectively.
2) There exists a path $\langle \text{SRC}, v_{A_1}, \dots, v_{A_k}, \text{SINK} \rangle$ in N if and only if there exists a tuple $\mathbf{u} \in R_\Omega$ such that we observe a sequence of annotations $(a_{A_1}, \dots, a_{A_k})$ in $\mathbf{u}$, where a_{A_i} is an annotation of type $A \in \Sigma$ and i is the position in the sequence.
3) Given an edge $e(v_{A_i}, v_{A_{i+1}}) \in E$, its capacity is the sum of all the annotations that would be preserved by selecting all the tuples $\mathbf{u} \in R_\Omega$ s.t. we can observe a sequence of annotations $(a_{A_1}, \dots, a_{A_i}, a_{A_{i+1}}, \dots, a_{A_k})$ in $\mathbf{u}$.

Paths on the network are possible attribute structures for R. Incorrect annotations lead to networks where multiple nodes are labelled with the same attribute. A path is *valid* if and only if it does not contain two nodes with the same attribute label.

Algorithm 2 constructs the network starting from an annotated relation. It iterates over all tuples (line 3) and their annotations (line 6), creating a new node whenever it observes a new annotation type or an existing one but in a different context. A suitable labelling function λ associates each node to a pair $\langle A, (A_1, \dots, A_k) \rangle$ denoting the type of the node and its context (line 11), context represented as the sequence of types observed in $\mathbf{u}$ *before* A (line 15). The weight w of an edge (v_i, v_{i+1}) is the total number of annotations we observe if we consider the sequence $(\text{SRC}, \dots, v_i, v_{i+1})$ (line 4).

The network has at most a number of nodes equal to the total number of annotations and a number of paths equal to the number of tuples in R_Ω. In the worst case, where each attribute has a different annotator, the number of annotations is $n \cdot \text{arity}(\Sigma)$, where n is the total number of tokens in R, i.e., each token of each tuple is annotated with all attributes. The network can therefore be constructed in polynomial time.

Algorithm 1 uses the computed network to compute the max-flow sequences S. For the max-flow computation, we adopted the Dinitz blocking flow algorithm to ease the implementation. We first compute augmenting paths with Dinitz's algorithm (line 9), then we select only valid paths, i.e., paths where there are not two nodes with the same attribute (lines 10–11). Dinitz's algorithm runs in $\mathcal{O}(|V|^2 \cdot |E|)$. The number of computed sequences depends on the number of tuples that are *covered* by each sequence. In particular, we iterate over the following steps: *(i)* Compute the max-flow sequence s in the network. *(ii)* Remove from R_Ω all annotations matching s or one of its subsequences and recompute the network (lines 12–15). The iteration continues until the number of tuples that are covered by some computed sequences exceeds

Algorithm 2: Flow Network.

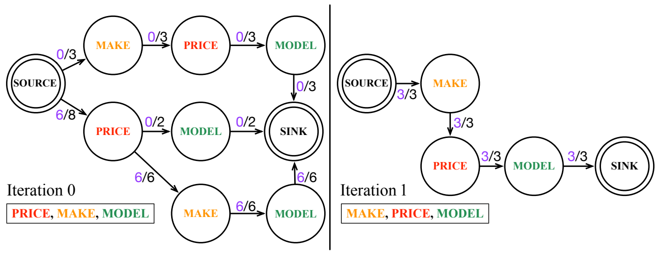
```
   input : $R_\Omega$ – annotated relation
   output: $\langle V, \lambda, E, w \rangle$ – flow network
1  $V \leftarrow \{\text{SRC}, \text{SINK}\}$;
2  $E, \lambda, w \leftarrow \emptyset$;
3  foreach $\mathbf{u} \in R_\Omega$ do
4      $ann\# \leftarrow |\text{anns}(\mathbf{u})|$;
5      $A_{seen} \leftarrow (\text{SRC})$; $v_{prev} \leftarrow \text{SRC}$;
6      foreach $\langle [i, j), A \rangle \in \text{anns}(\mathbf{u})$ do
7          $v \leftarrow \{v \in V \mid \lambda(v) = \langle A, A_{seen} \rangle\}$;
8          if $v = \text{null}$ then
9              $v \leftarrow$ new Node;
10             $V \leftarrow V \cup \{v\}$;
11             $\lambda \leftarrow \lambda \cup \{(v, \langle A, A_{seen} \rangle)\}$;
12             $E \leftarrow E \cup \{\langle v_{prev}, v \rangle\}$;
13             $w \leftarrow w \cup \{(\langle v_{prev}, v \rangle, 0)\}$;
14         $w(\langle v_{prev}, v \rangle) \leftarrow w(\langle v_{prev}, v \rangle) + ann\#$;
15         $A_{seen} \leftarrow (A_{seen}, A)$;// identify the context
16     if $\langle v_{prev}, \text{SINK} \rangle \notin E$ then
17         $E \leftarrow E \cup \{\langle v_{prev}, \text{SINK} \rangle\}$;
18         $w \leftarrow w \cup \{(\langle v_{prev}, \text{SINK} \rangle, 0)\}$;
19     $w(\langle v_{prev}, \text{SINK} \rangle) \leftarrow w(\langle v_{prev}, \text{SINK} \rangle) + ann\#$;
20 return $\langle V, \lambda, E, w \rangle$
```

a given threshold $t_{flow} \in [0, 1]$ (line 16). The output of the algorithm is a ranking of attribute sequences by number of covered tuples. The max-flows computation on a network aims to reconstruct *(i)* the original (valid) attribute sequence of the original website, *(ii)* that maximises the total number of preserved annotations (since the majority of annotations are to be considered correct).

Example 7. Consider the relation of Example 2, with target schema $\Sigma = \{\text{MAKE}, \text{MODEL}, \text{PRICE}\}$ and annotators $\Omega = \{\omega_{\text{MAKE}}, \omega_{\text{MODEL}}, \omega_{\text{PRICE}}\}$. The following is a possible annotated relation R_Ω, having ω_{MAKE} and ω_{MODEL} incomplete, i.e., they miss the values "Citroën" and "Allroad quattro".

u_1		£19k	Audi A3 Sportback
u_2		£43k	Audi A6 Allroad quattro
u_3	Citroën	£10k	C3
u_4	Ford	£22k	C-max Titanium X

The network is shown on the left (multiple nodes with same label are due to different contexts). The maximum flow is 6 corresponding to the sequence s_1: (PRICE, MAKE, MODEL) covering the first three tuples in R_Ω. Note that u_3 is also covered since it contains a subsequence of s_1. If we set $t_{flow} > 0.75$, the algorithm then deletes the annotations covered by s_1, leading to the network on the right. The sequence corresponding to the maximum flow of 3 is now s_2: (MAKE, PRICE, MODEL). The computation stops here as the set $\{s_1, s_2\}$ covers the entire relation.



A step-by-step example of the network construction process and the optimality w.r.t. maximum-likelihood sequences using memory-less Markov Chains can be inspected online [1].

The computation of max-flow sequences always terminates since *(i)* R_Ω is finite, and *(ii)* every time a max-flow sequence is computed, the corresponding annotations are removed from R_Ω, thus reducing the number of possible valid flows in the network. In the worst case, each tuple produces a new sequence, therefore the number of sequences is bounded by the size of the relation.

Algorithm 1 then removes from R_Ω all annotations that do not match any of the computed sequences (wrong annotations), leaving each tuple of R_Ω with only one annotation per attribute (lines 17–20). The output of the segmentation phase is the annotated relation R_Ω, without those annotations that disagree with the computed flow sequences S.

Induction. As a final step, the induction phase takes as input the modified annotated relation R_Ω and computes regular expressions segmenting attribute values. For each attribute $\mathrm{A} \in \Sigma$, the function `regexInduction` (line 23) produces a regular expression $\mathcal{E}_\mathrm{A}$ matching the substrings annotated as A in $\mathsf{T}(\mathbf{u})$, for each $\mathbf{u} \in \mathrm{R}_\Omega$. The expression $\mathcal{E}_\mathrm{A}$ is constructed by generalising the annotated spans.

The function `regexInduction` takes as input a set of (positive) examples: $\{\langle \mathsf{T}(\mathbf{u}_1), \mathsf{anns}(\mathbf{u}_1)\rangle, \ldots, \langle \mathsf{T}(\mathbf{u}_n), \mathsf{anns}(\mathbf{u}_n)\rangle\}$ where each example consists of the string representation of a tuple and the corresponding annotated spans. For each attribute A, the function takes the corresponding annotated spans in the examples and looks for *(i)* common length prefixes and or suffixes to the annotated content, or *(ii)* non-content strings that repeat a constant number of times. Non-content strings consist of punctuation, whitespace, special characters, and other *un-annotated* content. Among the computed expressions, the function returns the expression matching the highest number of examples, provided that this number exceeds a given threshold $t_{regex} \in [0,1]$. The algorithm also prefers expressions where the common prefix/suffix does not overlap with annotated content (i.e., it privileges robust expressions). When it fails to produce an expression matching enough examples, the function constructs an expression that matches the disjunction of the annotated values. We call this expression *value-based*. Value-based expressions are less robust since they depend on the observed attribute values. The method is an improved version of the text-range induction algorithm used in [18] (details are deferred to [1]). The output of the induction phase (and of the repair algorithm) is the set of pairs computed as described above, consisting of a Σ-repair of R w.r.t. Σ and Ω.

Example 8. Consider the relation and the set of max-flow sequences of Example 7. Algorithm 1 computes the following Σ-repair, with $\Pi = (1,2,3)$:

$$\langle \mathrm{MAKE}, \mathsf{value\text{-}based}(\$, \textit{'Audi} \mid \textit{Ford'})\rangle$$
$$\langle \mathrm{MODEL}, \mathsf{substring\text{-}after}(\mathsf{substring\text{-}after}(\$, \sqcup), \sqcup)\rangle$$
$$\langle \mathrm{PRICE}, \mathsf{substring\text{-}after}(\mathsf{substring\text{-}before}(\$, k\sqcup), \sqcup)\rangle$$

We use the value-based as a shortcut for a case statement in XPath. Note that the expressions for PRICE and MODEL are robust, provided that examples are representative of the corresponding domains. In contrast, the expression for MAKE is value-based (similar to the one computed in Example 6) and therefore much less robust. In this case, the result has a higher fitness than the one computed using directly the annotations (it would not identify "Allroad quattro" as belonging to MODEL).

The induction of a regular expression for an attribute A requires the iteration over all the examples and the inspection of tokens occurring before and after the annotated spans. Since we repeat this procedure for all the attributes, this requires $\mathcal{O}(n \cdot \mathsf{arity}(\Sigma))$ steps, with n the number of tokens in R_Ω.

Optimality. We now show that Algorithm 1 produces good approximations of joint repairs if certain conditions on the error rate of the annotators are satisfied.

Let $\langle \nu_1, \ldots, \nu_\mathrm{M}\rangle$ be the error rates for the annotators $\langle \omega_1, \ldots, \omega_\mathrm{M}\rangle$ in Ω. Moreover, let σ_Ω be the joint repair constructed by directly using the values returned by the annotators, and σ be the joint repair produced by Algorithm 1. The following result holds.

Theorem 3: For every relation R and wrapper W such that $\mathsf{W}(\mathsf{P}){=}\mathrm{R}$ for some collection of pages P, every schema Σ and every set of annotators Ω, if $\max_{\omega \in \Omega}(\nu_\omega) < (1 - \sum_{\omega \in \Omega} \nu_\omega)$, then $f(\sigma_\Omega(\mathrm{R}), \Sigma, \Omega) \leq f(\sigma(\mathrm{R}), \Sigma, \Omega)$.

In other words, the joint repair computed by Algorithm 1 will always induce a fitness that is not less than the one induced by a joint repair that directly uses the annotations, provided that the number of tuples annotated with correct sequences is more than the maximum number of errors made by an individual annotator on the same relation. Theorem 3 can also be seen as a precise quantification of how many errors can be made by an annotator before being considered systematic. We refer to the online appendix [1] for the proof.

V. Evaluation

Our repair approach is implemented in Java and SQL, and uses the ROSeAnn [4] entity recogniser framework to produce annotated relations. For this paper we have used all of the 11 annotators described in [4], which contains an extensive evaluation of the involved annotators. Our experimental evaluation aims at demonstrating *(i)* the independence of our approach w.r.t. domains and wrapper induction systems, *(ii)* the robustness w.r.t. annotation errors, *(iii)* an increased accuracy and performance w.r.t. redundancy-based methods.

Datasets. The dataset consists of 100 websites from 10 domains and is an enhanced version of SWDE [20], a benchmark commonly used in web data extraction. SWDE's data is sourced from 80 sites and 8 domains: auto, book, camera, job, movie, NBA player, restaurant, and university. For each website, SWDE provides collections of 400 to $2k$ *detail* pages (i.e., where each page corresponds to a single record). We complemented SWDE with collections of *listing* pages (i.e., pages with multiple records) from 20 websites of real estate (RE) and auto domains. The final dataset (Table I) has more than $120k$ pages, $130k$ records, and $500k$ attribute values. SWDE comes with ground-truth data created under the assumption that wrapper induction systems could only generate extraction rules with DOM-element granularity, i.e., without segmenting text nodes.

TABLE I: Dataset characteristics.

Domain	*Type*	*Sites*	*Pages*	*Records*	*Attributes*
Real Estate	listing	10	271	3,286	15
Auto	listing	10	153	1,749	27
Auto	detail	10	17,923	17,923	4
Book	detail	10	20,000	20,000	5
Camera	detail	10	5,258	5,258	3
Job	detail	10	20,000	20,000	4
Movie	detail	10	20,000	20,000	4
Nba Player	detail	10	4,405	4,405	4
Restaurant	detail	10	20,000	20,000	4
University	detail	10	16,705	16,705	4
Total	-	100	124,715	129,326	78

Since modern wrapper induction systems support text-node segmentation, we have refined the ground truth accordingly. As an example, in the camera domain, the original ground truth for MODEL consisted of the entire product title but the text node also provides, e.g., COLOUR, PIXELS, and MANUFACTURER.

Wrapper induction systems. We generated input relations for our evaluation using four wrapper induction systems: DIADEM [18], DEPTA [35] and ViNTs [38] for listing pages, and RoadRunner [11] for detail pages.[1] The output of DIADEM, DEPTA, and RoadRunner can be readily used in the evaluation. ViNTs, on the other hand, does not have the notion of attributes and only segments search listings into rows corresponding to records. We therefore post-processed its output, typing the content of lines from different records that are likely to have the same semantics. The distance metric used is equivalent to the ones used, e.g., by WEIR [3] and TEGRA [6].

Metrics. The performance of the repair is evaluated by comparing wrapper-generated relations against the SWDE ground truth before and after the repair. The metrics used for the evaluation are Precision, Recall, and F_1-Score computed at attribute level. Both the ground truth and the extracted values are normalised, and exact matching between the extracted values and the ground truth is required for a hit. Due to space limitations, we only present the most relevant results. The full evaluation, together with the dataset, gold standard, extracted relations, the code of the normaliser and of the scorer are available at [1]. All experiments are run on a desktop with an Intel quad-core i7 at 3.40GHz with 16 GB RAM.

Relation-level Accuracy. The first two questions we want to answer are: whether joint repairs are necessary and what their impact is in terms of quality. Table II reports, for each system, the percentage of: *(i)* Correctly extracted values. *(ii)* Under-segmentations, i.e., when values for an attribute are extracted together with values of other attributes or spurious content. *(iii)* Over-segmentations, i.e., when attribute values are split over multiple fields. As anticipated in Section II, this rarely happens since an attribute value often falls within a single text node. In this setting an attribute value can be over-segmented only if the extraction system is capable of splitting single text nodes (DIADEM), but even in this case the splitting happens only when the system can identify a strong regularity within the text node. *(iv)* Misplacements, i.e., values are placed or labeled as the wrong attribute. This is mostly due to lack of semantic knowledge or overlapping attribute domains. *(v)* Missing values, due to lack of regularity and optionality in the web source (RoadRunner, DEPTA, ViNTs) or

[1] RoadRunner configuration has been optimised for detail pages.

TABLE II: Wrapper induction errors.

System	*Correct (%)*	*Under Segmented (%)*	*Over Segmented (%)*	*Misplaced (%)*	*Missing (%)*
DIADEM	60.9	34.6	0	23.2	3.5
DEPTA	49.7	44	0	25.3	6
ViNTs	23.9	60.8	0	36.4	15.2
RoadRunner	46.3	42.8	0	18.6	10.4

missing values from the domain knowledge (DIADEM). Note that the numbers do not add up to 100% since errors may fall into multiple categories. These numbers clearly show that there is a quality problem in the produced relations and also support the assumption of atomic misplacements.

Figure 2 shows the impact of the joint repair on our metrics. Light (resp. dark)-coloured bars denote the quality of the relation before (resp. after) the repair.

A first conclusion that can be drawn is that repairs are always beneficial. From 697 extracted attributes, 588 (84.4%) require some form of repair, with 50% being the average pre-repair F_1-Score across systems. Our approach induces a correct regexes for 335 (57%) attributes, falling back to value-based expressions for the remaining 253 (43%). We repair at least one attribute in every wrapper and we repair more than 75% of attributes in more than 80% of the cases.

Among the considered systems, DIADEM delivers, on average, the highest pre-repair F_1-Score ($\geq$60%), but it never exceeds 65%. RoadRunner is on average worse than DIADEM but it reaches a better 70% F_1-Score on restaurant. Websites in this domain are in fact highly structured and individual attribute values are contained in a dedicated text node. When attributes are less structured, e.g., on book, camera, movie, RoadRunner has a significant drop in performance. ViNTs delivers the worst pre-cleaning results.

In terms of accuracy, our approach boosts F_1-Score between 15% and 60%. Performance is consistently close to or above 80% across domains and, across systems (except for ViNTs), with a peak of 91% for RoadRunner on NBA player.

The following are the remaining causes of errors: *(i)* Missing values cannot be repaired as our approach can only use the data available in the relation. This mostly affects RoadRunner and ViNTs. *(ii)* There are cases where the structure of the content is regular enough to induce (suboptimal) $f^{\Sigma}_{\text{XPATH}}$expressions without triggering the use of value-based expressions. This often results in a sub-optimal repair of the relation. This is more evident on domains such as book, camera, and movie where some of the attributes only occur in irregular positions in text summaries or titles. *(iii)* On very irregular relations (e.g., those generated by ViNTs), the repair essentially relies on the accuracy of the annotators.

Note that numbers published in this paper differ in parts from previously published results (e.g., [3], [11]). This is due to the use of *exact* matches (in this paper) instead of *containment*.

Another interesting question is whether the ability to segment text nodes during wrapper induction (as done, e.g., by DIADEM) has an impact on the quality of the repair. It turns out that the impact is very limited: if we disable text-node segmentation in DIADEM and re-run the experiment, the number of values requiring segmentations ranges from 4.7% in auto

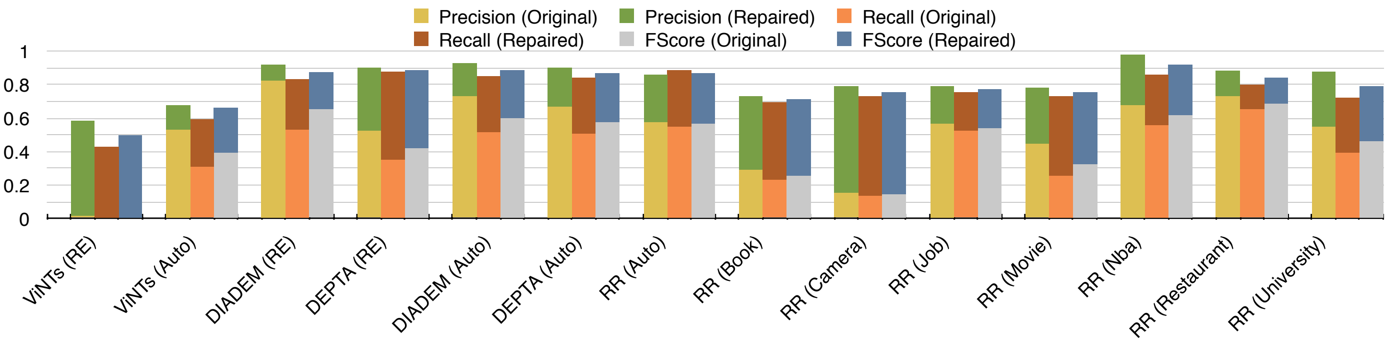


Fig. 2: Impact of repair.

to 30% in real estate, with an identical effect in almost all domains.

Attribute-level accuracy. Another question is whether there are substantial differences in attribute-level accuracy. The top of Table III shows attributes where the repair is very effective (F_1-Score$\simeq$1 after repair). These values appear as highly structured attributes on web pages and the corresponding expressions repair almost all tuples. As an example, DOOR NUMBER is almost always followed by suffixes *dr* or *door*. In these cases, the wrapper induction under-segmented the text due to lack of sufficient examples.

TABLE III: Attribute-level evaluation.

System	*Domain*	*Attribute*	*Original F_1-Score*	*Repaired F_1-Score*
DIADEM	real estate	POSTCODE	0.304	**0.947**
DIADEM	auto	DOOR NUMBER	0	**0.984**
DEPTA	real estate	BATHROOM NUMBER	0.314	**0.973**
DEPTA	auto	MAKE	0.564	**0.986**
DIADEM	real estate	CITY	0	0.59
DEPTA	real estate	COUNTY	0	0.728
DIADEM	auto	ENGINE TYPE	0	0.225
DEPTA	auto	PRICE	0.711	0.742

For attributes such as CITY and COUNTY, despite a significant boost (59% and 72% respectively) produced by the repair, the final F_1-Score is still low. These are irregularly structured attributes, often co-occurring with others, e.g., STATE, POSTCODE, in ways that cannot be easily isolated by regular expressions. Despite not having syntactic regularity, these attributes are *semantically* related, e.g., COUNTY is usually after CITY and before POSTCODE, and could be captured by extending $f^{\Sigma}_{\text{XPATH}}$ with NER capabilities [4].

An exceptional case is ENGINE TYPE, where the value *Petrol* is also recognised as COLOUR. This causes a loss of performance as it creates a systematic error in the annotated relation. Another exception is the case of PRICE in relations generated by DEPTA. DEPTA extracts large chunks of text with multiple prices among which the annotators cannot distinguish the target price reliably, resulting in worse performance.

Independent evaluation. We performed an extraction of restaurant chain locations in collaboration with a large social network, which provided us with 210 target websites. We used DIADEM as a wrapper induction system and we then applied joint repair on the generated relations. The accuracy has been manually evaluated by third-party rating teams on a sample of nearly 1,000 records of the 276,787 extracted. Table IV shows Precision and Recall computed on the sample (values higher than 0.9 are highlighted in bold). In order to estimate

TABLE IV: Accuracy of large scale evaluation.

Attribute	*Precision*	*Recall*	*% Modified values*
LOCALITY	**0.993**	**0.993**	11.34%
OPENING HOURS	**1.00**	0.461	17.14%
LOCATED WITHIN	**1.00**	0.224	29.75%
PHONE	**0.987**	0.849	50.74%
POSTCODE	**0.999**	**0.989**	9.4%
STREET ADDRESS	**0.983**	**0.98**	83.78%

the impact of the repair, we computed, for each attribute, the percentage of values that are different before and after the repair step. These numbers are shown in the last column of Table IV. Clearly, the repair is beneficial on all of the cases. For OPENING HOURS and LOCATED WITHIN, where recall is very low, the problem is due to the fact that these attributes were often not available on the source pages, thus being impossible to repair. The independent evaluation proved that our repair method can scale to hundreds of thousands of non-synthetic records. On the other hand, the joint repair is bound to the accuracy of the extraction system, i.e., it cannot repair data that has not been extracted.

We have previously shown (Section IV) that an optimal approximation of a joint repair can be computed efficiently. To stress the scalability of our method, we created a synthetic dataset by modifying two different variables: n—number of records, with an impact mostly on the induction of regular expressions, since it increases the number of examples; k—number of attributes, which influences the size of the flow network and the computation of the maximum flow. The synthetic relations are built to produce the worst case scenario, i.e., each record contains k annotated tokens, each annotation has a different context and each record produces a different path on the network. This results in a network with $n \cdot k + 2$ nodes, and $n \cdot k + n$ edges. The chart on the left of Figure 3 plots the running time over an increasing number of records (with number of attributes fixed), while the chart on the right

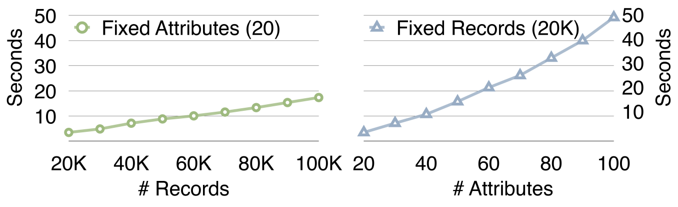


Fig. 3: Running time.

increases the number of attributes (with number of records fixed). As expected, the joint repair grows linearly w.r.t the size of the relation, and polynomially w.r.t. the number of attributes. In the extreme case, the computed network contains 10M nodes and 10.1M edges. The largest network obtained on non-synthetic datasets has 39,148 nodes and 45,797 edges (book), with repairs computed in less than 3 seconds.

Comparative evaluation. We compare our approach against WEIR [3], a wrapper induction and data integration system that can be used to compute a joint repair of a relation w.r.t. a schema. WEIR induces wrappers by generating candidate expressions using simple heuristics and by filtering them using instance-level redundancy across multiple web sources, i.e., it picks, among candidate rules, those that consistently match similar values on different sources. We compare with WEIR as the only other similar system, Turbo Syncer [8], is significantly older, and we were not able to obtain an implementation.

WEIR uses only redundant values for rules selection, resulting in relations with missing values (and records). We compared against WEIR on SWDE original dataset, the same one used in their evaluation [3] (using RoadRunner as extraction system). We evaluated WEIR and our approach in two separate settings: Figure 4 shows the performance of our approach and WEIR on each domain, computed on redundant records only, while in Figure 5 we also take into account non-redundant ones. A first observation is that redundant records are a small fraction of the whole relation, thus limiting the recall (shown on top of the bars in Figure 4). The results show that, if we limit the evaluation to redundant values only, our approach delivers same or better performance than WEIR. Interesting cases are auto, restaurant and university, where our approach outperforms WEIR by more than 10% in F_1-Score. In particular, WEIR suffers from false redundancy caused by a lax similarity measure and under-segmented text nodes. The only case where WEIR performs better than our approach is in movie, where the presence of multivalued attributes (such as GENRE) causes the selection of suboptimal max-flow sequences. If we also consider all values, including non redundant ones, our approach clearly outperforms WEIR in every domain, with a peak of 36% boost in F_1-Score in camera.

In terms of running time, WEIR requires an average of 30 minutes per domain whereas our approach repairs a domain in less than 2 minutes. This is due to the way WEIR exploits cross-source redundancy, i.e., instances in a source are compared against instances of all other sources. As a consequence, the running time increases with the number of sources. Our approach instead repairs each source in parallel.

We also run a preliminary comparative evaluation with

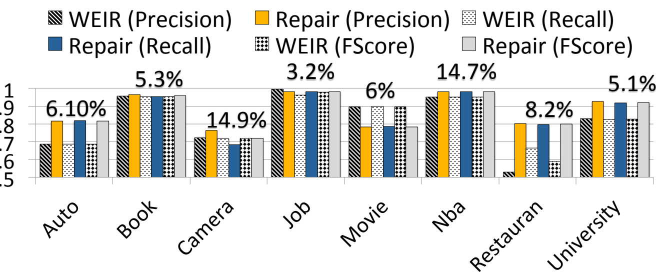


Fig. 4: Comparison with WEIR (Redundant values)

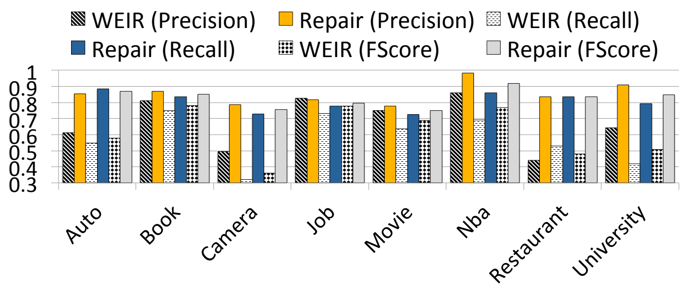


Fig. 5: Comparison with WEIR (all values)

Google Data Highlighter, a supervised data annotation tool that can be used to produce tabular data from web pages. A discussion is available at [1].

Ablation study. In this experiment we measured the impact of each phase of the joint repair computation on F_1-Score for the most relevant scenarios (we found similar results in other scenarios but those have not been included due to space reasons). With respect to Figure 6, original (or) refers to the

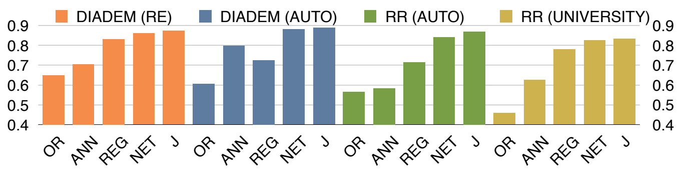


Fig. 6: Impact of individual components

original (i.e., before repair) quality of the relation, while joint (j) is the post-repair quality. annotator (ann) shows the effect of constructing the repair by directly using annotations, regex (reg) shows the performance when only regexes are induced (i.e., without value based expressions), network (net) shows the repair performance by using only value based expressions computed from max-flow sequences (i.e., no regex induction).

As we can see, the direct use of annotations for repair or regex induction delivers poor results. The major contribution to the quality is the use of max-flow sequences that uncover the underlying structure of the relation and eliminate noisy annotations. Regex induction is still beneficial afterwards to recover misses of the annotators. The most striking case is STREET_ADDRESS in the real estate domain. The attribute is hardly recognised by annotators (accuracy around 50%), however its structure in the relation is very regular and the after-repair accuracy reaches 85%.

Thresholding. This second experiment measures the effect of the thresholds t_{flow} and t_{regex} on performance. Figure 7 shows the variation of F_1-Score for the most interesting scenarios (other scenarios report similar results). The setting of excessively low thresholds negatively impacts the performance, as it causes a premature induction of regexes that repair only a small number of records. However, there are cases, e.g., DIADEM on real estate, where a lower threshold helps to recover misses of the annotators. In domains where attributes are better structured or the annotator is more accurate, e.g., auto, the best performance is achieved by setting a high threshold. Overall, the variation in performance is anyway limited ($\leq$3%) and the average best performance is obtained with a 75% threshold.

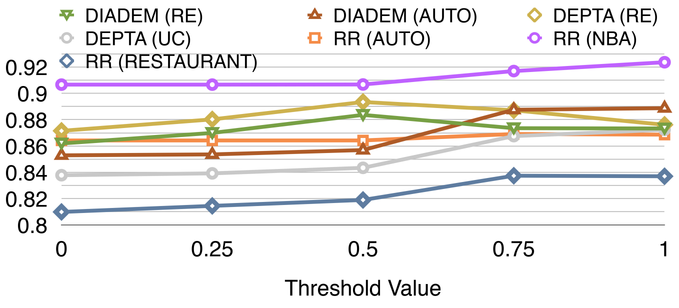


Fig. 7: Impact of t_{flow} and t_{regex} threshold

Effect of annotator accuracy. We gradually decreased the Recall of our annotators by randomly eliminating a number of annotations and observing the effect on F_1-Score while keeping a fixed regex induction threshold (0.75). To lower the effect of sampling bias, we ran the experiment 30 times with different annotation sets and took the average performance. The accuracy numbers are limited to those attributes where our approach induces regular expressions, since it is already clear that annotator errors directly reduce the accuracy of value-based expressions. This is still a significant number of attributes, i.e., $\geq$65% in all cases except for RoadRunner on book (35%), and RoadRunner on movie (46%). Figure 8 shows

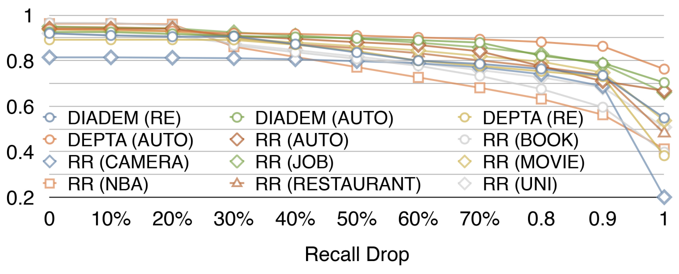


Fig. 8: Annotator recall drop - Fixed threshold

the impact of a drop in recall (x-axis) on F_1-Score. As we can see, our approach is robust to a drop in recall until we reach 80% loss, then the performance rapidly decays. This is somehow expected, since the regular expressions compensate for the missing recall up to the point where the max-flow sequences are no longer able to determine the underlying attribute structure reliably.

Figure 9 show the effect on F_1-Score if we set a low regex-induction threshold (i.e., 0.1) instead. Clearly, in this case our approach is highly robust to annotator inaccuracy and we notice a loss in performance only after 80-90% loss in recall. In summary, a lower regex-induction threshold is advisable when we know that annotators have low recall. Even involving an annotator with very low accuracy, our approach is robust

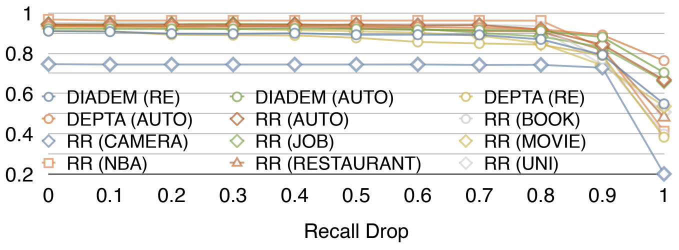


Fig. 9: F_1-Score variation with a threshold value of 0.1

enough to overcome the errors introduced by the annotator.

VI. Related Work

Computing joint repairs is one of the many *maintenance* problems faced in web data extraction [5], [22], [25], [27], [28]. However, classical wrapper maintenance has assumed perfect, typically human-created wrappers to begin with, with errors only being introduced over time due to change in the sources. When covering thousands or hundreds of thousands of sources with automatically or semi-supervised wrapper induction this assumption is no longer valid.

Closer in spirit to joint repairs are techniques to generate wrappers from background data [3], [8], [17], [36]. These techniques implicitly align background data and wrappers as part of the generation process. The closest works to ours are Turbo Syncer [8] and WEIR [3], which use instance-level redundancy across multiple sources to compute extraction rules on individual sites that, together, can be used to effectively learn wrappers without supervision. An advantageous side-effect of these approaches is the construction of "compatible" relations that can be more easily integrated. Differently from Turbo Syncer and WEIR, our approach assumes the existence of an already generated wrapper to be repaired w.r.t. a target schema. From a practical point of view, both Turbo Syncer and WEIR can be adapted to compute joint repairs, however, as shown in Section V with a significantly worse performance than our approach due to their reliance on redundancy. Our approach also eliminates the need for re-induction of wrappers, leading to better runtime performance.

Redundancy. Instance-level redundancy across web sources has been previously used in different contexts to detect and repair inconsistent data extracted from the web [3], [7], [8], [13]. Redundancy-based approaches face two main obstacles: *(i)* it is not always possible to leverage sufficient redundancy in every domain, (see, e.g., the number of redundant records in SWDE of Figure 4), and *(ii)* redundancy-based methods require access to a substantial number of sources that have, so far, limited their *scalability* (see, e.g., WEIR running time). Encoding the redundancy by other means, e.g., through entity recognisers and knowledge bases, has proven beneficial to circumvent the scalability problems without sacrificing generality of the approaches [7], [18]. Our approach achieves this via an ensemble of entity recognisers [4], some of which are trained using redundancy-based methods.

Cleaning, segmentation and alignment. Traditional data cleaning methods focus on the detection and repair of database inconsistencies, using, e.g., statistical value distributions [31], [34], constraints [2], [14], [16], [29], and knowledge bases [7]. Differently from our setting, cleaning methods operate on relation(s) that contain incorrect values but are assumed to be correctly segmented.

A more relevant body of work is list segmentation/table induction techniques, targeting the induction of structured records from unstructured (i.e., wrongly segmented) lists of values. These are alternatives to the segmentation based on flow networks used in our approach. Being inspired by tagging problems common in bio-informatics and other areas, these approaches traditionally require some form of supervision. Many require an initial seed of correctly segmented records [10],

[21], [23], [26], [37], while others require positive and negative examples of valid field/column values as training data [24], [32], sometimes leveraging existing knowledge bases [9], [30] or, again, instance-level redundancy [6], [13].

While Google Data Highlighter relies on Freebase and therefore targets only a few entity types [30], an unsupervised method close to our segmentation strategy is employed in TEGRA [6]. Differently from, e.g., List Extract [13], TEGRA looks at the relation globally, circumventing the intractability of the problem by constructing approximated but optimal solutions. The key difference between TEGRA and our approach is in the distance metrics used for alignment. TEGRA's is essentially equivalent to cross-site instance-level redundancy (carrying all its pros and cons), while our approach encodes it via attribute oracles. Another major difference of our setting is the presence of misplacements and spurious content in the lists to be segmented. It is worth mentioning that the problem of segmenting lists with spurious content was already tackled by [33] but, again, using cross-site, instance-level redundancy.

VII. Conclusion

This paper studies the problem of computing joint repairs for web wrappers and generated relations, proposing an unsupervised framework that is independent on the domain and of the wrapper induction system. A future direction is to extend our approach to multivalued attributes, by allowing sequences to have multiple nodes with the same attribute label. Another interesting direction is to use the repairs to incrementally learn entity recognisers: both in terms of precision (identification of wrong annotations), and in terms of recall (discovery of new attribute values through regex generalisation).

References

[1] "Online Appendix," http://diadem.cs.ox.ac.uk/wadar.

[2] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi, "A cost-based model and effective heuristic for repairing constraints by value modification," in *SIGMOD*, 2005, pp. 143–154.

[3] M. Bronzi, V. Crescenzi, P. Merialdo, and P. Papotti, "Extraction and integration of partially overlapping web sources," *VLDB*, vol. 6, no. 10, pp. 805–816, 2013.

[4] L. Chen, S. Ortona, G. Orsi, and M. Benedikt, "Aggregating semantic annotators," *PVLDB*, vol. 6, no. 13, pp. 1486–1497, 2013.

[5] B. Chidlovskii, B. Roustant, and M. Brette, "Documentum eci self-repairing wrappers: Performance analysis," in *SIGMOD*, 2006, pp. 708–717.

[6] X. Chu, Y. He, K. Chakrabarti, and K. Ganjam, "TEGRA: Table extraction by global record alignment," in *SIGMOD*, 2015, pp. 1713–1728.

[7] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. Ye, "Katara: A data cleaning system powered by knowledge bases and crowdsourcing," in *SIGMOD*, 2015, pp. 1247–1261.

[8] S.-L. Chuang, K. C.-C. Chang, and C. Zhai, "Context-aware wrapping: synchronized data extraction," in *VLDB*, 2007, pp. 699–710.

[9] E. Cortez, A. S. da Silva, M. A. Gonçalves, and E. S. de Moura, "Ondux: On-demand unsupervised learning for information extraction," in *SIGMOD*, 2010, pp. 807–818.

[10] E. Cortez, D. Oliveira, A. S. da Silva, E. S. de Moura, and A. H. Laender, "Joint unsupervised structure discovery and information extraction," in *SIGMOD*, 2011, pp. 541–552.

[11] V. Crescenzi and P. Merialdo, "Wrapper inference for ambiguous web pages," *Applied Artificial Intelligence*, vol. 22, no. 1-2, pp. 21–52, 2008.

[12] N. N. Dalvi, R. Kumar, and M. A. Soliman, "Automatic wrappers for large scale web extraction," *PVLDB*, vol. 4, no. 4, pp. 219–230, 2011.

[13] H. Elmeleegy, J. Madhavan, and A. Halevy, "Harvesting relational tables from lists on the web," *VLDB J.*, vol. 20, no. 2, pp. 209–226, 2011.

[14] R. Fagin, B. Kimelfeld, F. Reiss, and S. Vansummeren, "Cleaning inconsistencies in information extraction via prioritized repairs," in *PODS*, 2014, pp. 164–175.

[15] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu, "Interaction between record matching and data repairing," in *SIGMOD*, 2011, pp. 469–480.

[16] F. Geerts, G. Mecca, P. Papotti, and D. Santoro, "The LLUNATIC data-cleaning framework," *PVLDB*, vol. 6, no. 9, pp. 625–636, 2013.

[17] A. L. Gentile, Z. Zhang, I. Augenstein, and F. Ciravegna, "Unsupervised wrapper induction using linked data," in *K-CAP*, 2013, pp. 41–48.

[18] G. Gottlob, T. Furche, G. Grasso, X. Guo, G. Orsi, C. Schallhart, and C. Wang, "Diadem: Thousands of websites to a single database," *PVLDB*, vol. 7, no. 14, pp. 1845–1856, 2014.

[19] F. E. Grubbs, "An introduction to probability theory and its applications," *Technometrics*, vol. 9, no. 2, pp. 342–342, 1967.

[20] Q. Hao, R. Cai, Y. Pang, and L. Zhang, "From one tree to a forest: a unified solution for structured web data extraction," in *SIGIR*, 2011, pp. 775–784.

[21] A. Kannan, I. E. Givoni, R. Agrawal, and A. Fuxman, "Matching unstructured product offers to structured product specifications," in *SIGKDD*, 2011, pp. 404–412.

[22] K. Lerman, S. N. Minton, and C. A. Knoblock, "Wrapper maintenance: A machine learning approach," *J. Artif. Int. Res.*, vol. 18, no. 1, pp. 149–181, 2003.

[23] A. Machanavajjhala, A. S. Iyer, P. Bohannon, and S. Merugu, "Collective extraction from heterogeneous web lists," in *WSDM*, 2011, pp. 445–454.

[24] I. R. Mansuri and S. Sarawagi, "Integrating unstructured data into relational databases," in *ICDE*. IEEE, 2006, pp. 29–29.

[25] X. Meng, D. Hu, and C. Li, "Schema-guided wrapper maintenance for web-data extraction," in *WIDM*, 2003, pp. 1–8.

[26] M. Michelson and C. A. Knoblock, "Creating relational data from unstructured and ungrammatical data sources," *J. Artif. Int. Res.*, vol. 31, no. 1, pp. 543–590, 2008.

[27] A. G. Parameswaran, N. N. Dalvi, H. Garcia-Molina, and R. Rastogi, "Optimal schemes for robust web extraction," *PVLDB*, vol. 4, no. 11, pp. 980–991, 2011.

[28] J. Raposo, A. Pan, M. Álvarez, and J. Hidalgo, "Automatically maintaining wrappers for semi-structured web sources," *Data Know. Eng.*, vol. 61, no. 2, pp. 331–358, 2007.

[29] S. Song, H. Cheng, J. X. Yu, and L. Chen, "Repairing vertex labels under neighborhood constraints," *PVLDB*, vol. 7, no. 11, pp. 987–998, 2014.

[30] P. Venetis, A. Halevy, J. Madhavan, M. Paşca, W. Shen, F. Wu, G. Miao, and C. Wu, "Recovering semantics of tables on the web," *PVLDB*, vol. 4, no. 9, pp. 528–538, 2011.

[31] D. Z. Wang, M. J. Franklin, M. Garofalakis, J. M. Hellerstein, and M. L. Wick, "Hybrid in-database inference for declarative information extraction," in *SIGMOD*. ACM, 2011, pp. 517–528.

[32] M. Wick, A. Culotta, and A. McCallum, "Learning field compatibilities to extract database records from unstructured text," in *EMNLP*, 2006, pp. 603–611.

[33] T.-L. Wong, W. Lam, and T.-S. Wong, "An unsupervised framework for extracting and normalizing product attributes from multiple web sites," in *SIGIR*, 2008, pp. 35–42.

[34] M. Yakout, L. Berti-Équille, and A. K. Elmagarmid, "Don't be scared: use scalable automatic repairing with maximal likelihood and bounded changes," in *SIGMOD*. ACM, 2013, pp. 553–564.

[35] Y. Zhai and B. Liu, "Web data extraction based on partial tree alignment," in *WWW*, 2005, pp. 76–85.

[36] ——, "Extracting web data using instance-based learning," *World Wide Web*, vol. 10, no. 2, pp. 113–132, 2007.

[37] C. Zhao, J. Mahmud, and I. Ramakrishnan, "Exploiting structured reference data for unsupervised text segmentation with conditional random fields." in *SDM*, 2008, pp. 420–431.

[38] H. Zhao, W. Meng, Z. Wu, V. Raghavan, and C. Yu, "Fully automatic wrapper generation for search engines," in *WWW*, 2005, pp. 66–75.