UNIVERSITÀ DEGLI STUDI DI ROMA TRE
**Dipartimento di Informatica e Automazione**
Via della Vasca Navale, 84 – 00146 Roma, Italy.

# Exploiting Information Redundancy to Extract and Integrate Data from the Web

LORENZO BLANCO†, MIRKO BRONZI†, VALTER CRESCENZI†, PAOLO MERIALDO†, PAOLO PAPOTTI†

**RT-DIA-151-2009** **June 2009**

† Dipartimento di Informatica e Automazione
Università di Roma Tre
00146 Roma – Italy

http://www.dia.uniroma3.it

**ABSTRACT**

An increasing number of web sites publish pages containing structured information about recognizable concepts, relevant to specific application domains - such as finance, sport, products. Although such information is spread across a myriad of sources, the web scale implies a relevant redundancy, which is apparent both at the intensional level (many sources share a core set of attributes), and at the extensional level (many instances occur in a number of sources). The paper investigates novel, domain independent techniques that exploit the redundancy of information among these sources to automatically extract and integrate data from the Web. The presented techniques have been implemented in a working prototype, which has been used to conduct a number of experiments on both synthetic and real-life web sites. Experimental results confirm the feasibility of the approach.

# Contents



# List of Figures

# 1 Introduction


Figure 1: Three web pages containing data about stock quotes from Yahoo! Finance, CNN Money, Google Finance

An increasing number of web sites deliver pages containing structured information about recognizable concepts, relevant to specific application domains - such as finance, sport, products. Consider for example the web pages shown in Figure 1, which contain information about stock quotes. Although human readers easily recognize such entities, current search engines are limited in exploiting the data offered by these sources. The development of scalable techniques to enable search engines to extract and integrate data from fairly structured large corpora available on the Web has been recently recognized as one of the utmost issue to wring out value from web data [1].

Extraction and integration of data from the Web involves two main activities: generating wrappers to extract data from pages, and defining semantic mappings among the data provided by the wrappers. Because of the web scale, these activities should be accomplished automatically, i.e. they cannot involve any human intervention and they cannot rely on domain specific solutions.

To cope with the complexity and the heterogeneity of web data, state-of-art approaches for the automatic extraction and integration of web data concentrate on information organized according to specific patterns that frequently occur on the Web. Meaningful examples are WebTables [11], which focuses on data published in HTML tables, and information extraction systems, such as TextRunner [4], which exploit lexical-syntactic patterns. As noticed in [11], even if a small fraction of the Web is organized according to these patterns, due to the web scale the amount of the involved data is impressive. It is worth observing that each pattern offers advantages and disadvantages. For example, WebTables is able to extract data with schemas that contain several attributes; however, these data have to be organized into HTML tables in order to be extracted. On the other side, information extraction systems extract and integrate huge amounts of data, but the schemas of the relations that they produce are limited, as they are able to manage only facts, i.e. simple binary relations.

This paper proposes a novel technique for the automatic extraction and integration of web data, that aims at exploiting an unexplored publishing pattern that we experienced frequently occur in data intensive web sites: large amounts of data are usually offered by pages that encode one flat tuple for each page. To give an example, consider Figure 2, which shows two pages from Yahoo! Finance; each page, and likely all the others for the many stocks that the site considers, is about a stock for a company and reports several attributes for it (e.g., volume, last trade, etc). We can abstract this representation and say that each web page displays a tuple, and that the whole collection of stock quote pages from that site corresponds to a relation, the "StockQuotes" relation. Interestingly, also the web sites for pages in Figure 1 expose their own "StockQuotes" relation.

It is easy to experience that for many disparate real world entities—going from stock quotes to athletes, from travels to movies—there are a lot of web sites that deliver their data following this publishing strategy: they offer collections of pages that can be though as encodings of a relation.

We have recently developed a crawling algorithm [6] that exploits the publishing strategy described above: it takes as input a small set of pages containing sample instances of a real world entity of interest, and then it explores the Web and gathers collections of pages from sites that deliver large amounts of data about the same entity of the seed pages. For example, if the crawler is fed with the three pages in Figure 1, it returns collections of pages, from many different web sites, that contain detailed data about stock quotes. Other methods for harnessing these collections can be developed by means of crawling techniques for the so-called hidden Web (such as those described in [23]), which is particulary rich of data.

The availability of large amount of data extracted and integrated from the Web enables countless applications. For example, existing search engines greatly benefit from accessing structured data repositories as they already offer data as results of keyword searches for a limited set of domains (e.g., for stock quote symbols like GOOG), while

independent approaches have been explored only recently.[1] Our techniques enable such possibilities, and can be proposed as a solution for search engines willing to automatically build and index structured repositories of disparate domains.

As another example of application, we have already studied the possibility to use integrated web data for evaluating the quality of the data and the trustiness of the web sources [7].

**Contributions** The contributions of the paper are the following. First, we introduce a domain independent and unsupervisioned algorithm to extract and integrate data from the Web at a large scale. Our proposal builds on the observation that although structured information is spread across a myriad of sources, the web scale implies a relevant redundancy, which is apparent both at the intensional level (many sources share a core set of attributes), and at the extensional level (many instances occur in a number of sources). Our framework takes advantage of the coupling of the wrapper inference and the data integration tasks, and, to the best of our knowledge, it is the first attempt to face both issues contextually.

Our second contribution is an instance based schema matching algorithm which is adaptive to the actual data. This algorithm presents significant advantages in general settings where multiple sources have to be matched.

Finally, the techniques described in this paper has been implemented in a working prototype, and integrated in a system, called Flint[2], that also includes the crawler to gather pages [6]; the system has been used to conduct a large set of experiments to show the feasibility of the techniques using both synthetic data and real-world data of different domains from a large number of web sites.

**Paper Outline** In the next Section we introduce an example to illustrate important observations that motivate our solution, and present an overview of our approach. The technical development of our extraction and integration techniques are presented in Sections 3, 4, 5. Section 6 reports the results of the experimental activity that we have conducted to evaluate our approach. Section 7 discusses related works. Section 8 concludes the paper.

# 2 Overview

Figure 1 presents three web pages, from different web sites. Each page contains structured information about a stock quote. In a database perspective, we may say that each page contains the data of one tuple from a "StockQuote" relation. Each source exposes data from its own relation; nevertheless, although some attributes are peculiar of a few sources—for example, the *Prev. close* is reported only by the first and second sources (Yahoo! Finance and CNN Money), whereas only the first source (Yahoo! Finance) publishes the *1y Target Est*—it is interesting to observe that there is a large set of attributes that are present in all the sources. For example, all the three sources publish the company name, current price, market capitalization, volume and average volume. This is a form of information redundancy that occurs at the schema level: there is a set of core attributes that are shared among different sources [20].

Figure 2 reports two more pages from Yahoo! Finance: each of them contains again data about one stock quote. Observe that pages with information about the same instances are published by many other web sites: for example, Figure 1 shows pages describing the Cisco Systems Inc. and the Microsoft Corporation stock quotes in the CNN Money and Google Finance web sites, respectively. The example clarifies another important observation about the abundance of information on the Web: as many sites publish information about the same object, on the Web there is a high degree of information redundancy also at the instance level. As a consequence, the values of common attributes, although coming from distinct sources, usually overlap whenever they refer to the same object.

In Figure 2 we also observe that regularities occur in the organization of data in Web pages from the same source. Notice that, although the two pages refer to different stock quotes, they share the same HTML template. Indeed, Yahoo! Finance publishes hundreds of pages like these: each page embeds into an HTML template the data about one stock quote tuple. Similarly, Google Finance and CNN Money deliver their stock quote pages by organizing the data in their own HTML templates. Another interesting observation from our examples is that the templates usually contain labels that explain the meaning of the published data to the final user. This is apparent in the pages of Figures 1 and 2.

To summarize, we observe that the web offers hundreds of sources providing pages that physically organize in a HTML template detailed information about instances of a given conceptual entity. At the intensional level, a core set of attributes is present in all the sources, while some attributes appear only in a restricted number of sources. At the

[1] See for example: http://www.google.com/squared

[2] http://flint.dia.uniroma3.it


Figure 2: Two Web pages about stock quotes from Yahoo! Finance

extensional level, many instances occur in several different sources, implying a strong overlap of data values over the shared attributes.

Our techniques to extract and integrate data from the Web build on these properties. We leverage the regularities that occur among pages from the same source to automatically infer a wrapper. The wrapper allows us to abstract from the physical organization of data into HTML pages: each page can be considered as a tuple and each source as a relation. Then, we exploit the redundancy of information to infer matchings between the attributes exposed by the sources (through the wrappers) and to improve the quality of the inferred wrappers.

In the remaining of this section, we first introduce notation and terminology; then, we formulate the problem addressed in this paper; finally we illustrate an overview of the proposed solution, which is further developed in the successive sections.

## 2.1 Problem statement

In our framework, a data source $S$ is a collection of pages from the same web site, $S = \{p_1, p_2, \ldots, p_n\}$, such that each page publishes information about one instance of a real world entity of interest. Pages in Figure 1 belong to three different sources (Yahoo! Finance, CNN Money, Google Finance) for the stock quote entity.

A wrapper $w$ is an ordered set of extraction rules, $w = \{er_1, er_2, \ldots, er_k\}$, that apply over a web page: each rule extracts a (possibly empty) string from the HTML of the page. We denote $er(p)$ the string returned by the application of the rule $er$ over the page $p$. The application of a wrapper $w$ over a page $p$, denoted $w(p)$, returns a tuple $t = \langle er_1(p), er_2(p), \ldots, er_k(p)\rangle$; therefore, the application of a wrapper over the set of pages of a source $S$, denoted $w(S)$, returns a relation $r = w(S)$, whose schema $R$ has as many attributes as the number of extraction rules of the wrapper: $R = (A_1, A_2, ..., A_k)$.

We say that two (or more) attributes are *distinct* if they come from different sources. A *mapping* $m$ is a set of distinct attributes with the same semantics.

Given a set of sources $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$, our goal is to infer:

- a set of wrappers $\mathcal{W} = \{w_1, w_2, \ldots, w_m\}$ (one per source) that extract a set of relations $r_1 = w_1(S_1), r_2 = w_2(S_2), \ldots r_m = w_m(S_m)$ from the sources;
- a set of mappings $\mathcal{M} = \{m_1, \ldots, m_k\}$ that correlate the attributes of the relations $r_1, r_2, \ldots r_m$.
- a ranked list of labels $\mathcal{L}_i$ for each mapping $m_i$, $m_i \in \mathcal{M}$.

## 2.2 Data Extraction and Integration

We now provide an overview of our solution to the above problem. In an initialization step, for every source a *bootstrapping wrapper* is automatically generated by leveraging on the local regularities of the page structure. The application of the wrappers over the sources returns a relation for each source. Since the process is completely automatic, at this stage we cannot rely on any feedback about the correctness of the wrappers, and the extracted relations are "opaque", i.e. their attributes are not associated with any semantic label. A key feature of our approach is the exploitation of the redundancy of data among the sources to infer mappings, to refine the wrappers, and to extract semantic labels for the mappings.

Our approach consists of an iterative process, which is initialized by creating a set of singleton mappings from the attributes of one of the extracted relations. Then, the process iterates over the remaining relations. For each relation the process evaluates whether its attributes can be added to some existing mapping. For each attribute of the current relation, the process compares its values with those of the attributes that already participate the mappings. Attributes with values that nicely match with those of an existing mapping are added to that mapping. The rationale is that if different attributes exhibit the same values for a number of tuples that refer to the same objects, it is likely that those attributes represent the same property. Attributes that are not added to any mapping originate new singleton mappings.

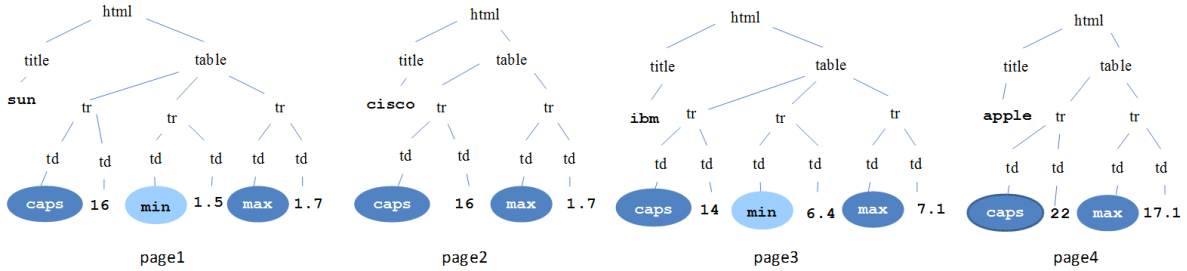


Figure 3: DOM trees of four stock quote pages

The matching process is rather articulated. On the one hand, as we cannot expect that sources are perfectly consistent, we consider "soft matches", which tolerate small imprecisions between the compared values. On the other hand, the tolerance degree cannot be too large, because it could lead to the creation of mappings that aggregate heterogeneous attributes. Our matching process dynamically establishes the tolerance degree to be applied on the matches by means of an effective technique that considers natural constraints that hold in our context.

The matching process influences also the refinement of the wrappers. Observe that whenever a match is partial, i.e. only a subset of the values overlap, either the involved attributes have different semantics (which would justify the different values), or the corresponding extraction rule of the bootstrapping wrapper is imprecise. The presence of partial matchings is then exploited to refine the bootstrapping wrappers. Alternative extraction rules are generated for attributes that partially match with a mapping. If a new extraction rule produces a better match, the wrapper is updated by replacing the weak rule with the new one. The new extraction rules produce different values: these values can improve the existing mappings, or they can give rise to new mappings. This is a key contribution of our approach: the redundancy of information in the mappings is used to refine the bootstrapping wrappers; dually, wrapper refinements lead to improve the mappings. As an interesting side effect, the wrapper refinement process associates the extraction rules with labels taken from the page template: typically these labels represent meaningful descriptions for the extracted values and can thus provide semantics to the inferred mappings.

In the following, we introduce the details of the process: Section 3 describes the generation of the bootstrapping wrappers; Sections 4 and 5 illustrate the algorithms to infer the mappings and to refine the wrappers.

# 3 Bootstrapping the Wrapper Generation

To infer the bootstrapping wrappers, we use a simple yet effective non supervisioned technique. For each source, we infer an initial wrapper by exploiting the local regularities that occur in the page structure, following some intuitions developed in [3].

In our framework, each page of a given source can be considered as an HTML encoding of a flat tuple. In this perspective given a set of pages from the same source, a wrapper is expected to extract the relation used to generate the pages. Therefore each rule in the wrapper should extract the values of the same attribute for every page.

We represent pages by means of their associated DOM trees, and we use XPath to express the extraction rules. Figure 3 shows a representation for the DOM trees of four pages belonging to a fictional source in the finance domain.

To describe our wrapper inference technique, it is convenient to abstract the page generation process as a procedure that fills the placeholders of an HTML template with the values of a tuple.[3] According to this model, within each source, pages generated by distinct tuples share all the elements that belong to the template, while they differ in the elements that correspond to attribute values. A wrapper could then be inferred by computing an XPath expression for each leaf node that does not belong to the template. Since elements of the template are shared by all the pages, they can be identified as those that are present in all the pages of the source.

However, the page generation model is further complicated by the possible presence of nullable attributes in the generating tuple. The publishing of null values can be based on two alternative strategies: $(i)$ the template placeholder is filled with an empty string; $(ii)$ a small part of the template, which is devoted to format the nullable attribute values, is not generated. For example, in the second and fourth page of Figure 3, the minimum price is not published and neither its (null) values, nor the corresponding formatting tags are reported in those pages.

With these ideas in mind, given a set of pages generated by the same template, DOM tree elements that occur exactly once in almost every page are considered as part of the template,[4] and they are called *invariant*. The gray nodes in the DOM trees depicted in Figure 3 represent invariant nodes: some of them appear always once (`caps`, `max`), while others, which are related to the presence of a null, might be present only in a subset of the pages (`min`).

All the leaf nodes that do not belong to the template are likely to represent values of the encoded tuple; therefore, for each of these nodes we compute an XPath expression. For simplicity, we compute absolute expressions, i.e. XPath expressions that specify the full path, including node positions, from the root to the leaf node. To give an example, the pages of Figure 3 would lead to the expressions reported in Figure 4.

$er_{A1} \leftarrow$ /html[1]/title[1]
$er_{A2} \leftarrow$ /html[1]/table[1]/tr[1]/td[2]
$er_{A3} \leftarrow$ /html[1]/table[1]/tr[2]/td[2]
$er_{A4} \leftarrow$ /html[1]/table[1]/tr[3]/td[2]

Figure 4: Extraction rules as XPath absolute expressions for the pages of Figure 3

The presence of null values produces irregularities in the pages that can affect the correctness of the inferred wrapper. In particular, some irregularities can lead to the inference of *weak* rules, that is, rules that do not correctly extract data for all the sample pages. For example, consider again the pages in Figure 3 and the corresponding inferred XPath expressions reported in Figure 4, and note that the second and fourth pages do not publish the minimum price. Figure 5 reports the relation extracted by these rules and shows that, because of the missing value, the extraction rules $er_{A3}$ extracts heterogeneous values: some of the values correspond to the maximum price ($cisco$ and $apple$), others to the minimum price.

It is important to observe that at this stage there is not enough information to evaluate the correctness of the wrapper. Section 5 describes how we leverage the abundance of redundant information among different sources to finalize the wrapper by replacing weak rules with alternative expressions, which extract values correctly.

# 4 Source Integration

The set of relations obtained by applying the bootstrapping wrappers over the input sources are analyzed with the

[3] The page generation model applies both for static or dynamically generated pages.

[4] We consider as template nodes the values occurring exactly once for a sufficient fraction $s = 1/3$ of the input pages.

| | $A_1$ | $A_2$ | $A_3$ | $A_4$ |
|---|---|---|---|---|
| page 1 | $sun$ | 16 | 1.5 | 1.7 |
| page 2 | $cisco$ | 16 | 1.7 | |
| page 3 | $ibm$ | 14 | 6.4 | 7.1 |
| page 4 | $apple$ | 22 | 17.1 | |

Figure 5: The relation extracted by the extraction rules in Figure 4 from the pages in Figure 3

twofold goal of *(i)* inferring mappings among their attributes, and *(ii)* refining the generation of the wrappers. The two activities are mutually dependent: the mapping inference process provides feedback about the quality of the wrappers; the wrapper refinement enforces the quality of the inferred mappings. We now describe how we address the issue of inferring mappings among the attributes of the relations provided by the bootstrapping wrappers; Section 5 illustrates the wrapper refinement process.

Our algorithm for inferring mappings is initialized by building one singleton mapping for each attribute of one of the input relations. Each mapping $m$ is associated with a *matching threshold*, denoted $t_m$, which is initialized to a default value $T$. Given an attribute $A$, and a mapping $m$, we call the *matching score* of $A$ against $m$, denoted $score(A, m)$, the similarity between the values of $A$, and those of the attributes in $m$. The similarity between two attributes $A$ and $B$, denoted $sim(A, B)$, is computed by comparing the values that they assume in a small sample set of *aligned* tuple pairs; two tuples are aligned if they refer to the same real world object. An attribute $A$ can participate a mapping $m$ if the matching score $score(A, m)$ is greater than the matching threshold $t_m$ of $m$. For the sake of presentation, we shall detail later, in Section 4.2, how the attribute similarity and the matching score are computed.

Once the initial set of singleton mappings is created, the algorithm iterates over the other relations. For each relation it computes the matching score of each attribute against all the existing mappings. If the matching score of an attribute $A$ against a mapping $m$ is greater than the matching threshold of $m$, then $A$ is added to $m$. If an attribute cannot match with none of the existing mappings, it gives rise to a new singleton mapping, and its matching threshold is assigned the default value $T$.

It is important to observe that an attribute might not have matched with any mapping because of the weakness of its corresponding extraction rule. For example, the rule $er_{A3}$ in Figure 4 produces wrong data for some of the pages it is applied on (namely, page 2 and page 4). The extracted data can thus match only partially with those of an existing mapping. At this stage, there is enough information to evaluate and possibly correct the extraction rule. Therefore each iteration concludes with the refinement of the wrapper used to generate the current relation. This task, which is described in Section 5, aims at improving the wrapper's extraction rules by using the values of the attributes in the mappings as a feedback to evaluate the precision of the rules.

It is worth observing that the number of singleton mappings can increase significantly for each iteration. To overcome this issue, we adopt the simple optimization of erasing mappings that remain singleton for a fixed number of iterations over the processed relations. The rationale is that in a large number of sources with redundant data, if an attribute does not match with any other attribute of a significant number of sources, it is likely that it will not match with any attribute of the whole collection. The algorithm concludes when all the relations have been processed.

The described approach, that we call *Naive Matching*, is limited by the strong dependence on the value of the matching threshold. High values of the threshold tend to generate many small mappings, because small imprecisions are not tolerated. Conversely, low values produce large heterogeneous mappings, composed by attributes with different semantics.

In our setting, it is not easy to set a threshold that represents a nice tradeoff between precision and recall. For example, in the finance domain, there are attributes, such as the minimum and the maximum price of a stock, that are quite similar. The threshold should be rather high if we do not want to confuse these attributes. On the contrary there are attributes, such as the volume, for which different approximations are expressed by distinct sources, and therefore a lower, more tolerant, threshold should be preferred. To address these issues we have developed a clever matching algorithm, called *SplitAndMerge*, that dynamically computes the matching threshold for every mapping.

## 4.1 SplitAndMerge Matching

The *SplitAndMerge* algorithm is based on the observation that it is unlikely that a source publishes the same attribute more than once, with different values. We can therefore assume that not identical attributes from the same source have always a different semantics. As a consequence, *SplitAndMerge* imposes the constraint, called the *SplitAndMerge* constraint, that a mapping is not allowed to include more than one attribute from the same source.

In *SplitAndMerge*, mappings are created iterating over the source relations as in the naive approach. However, before adding an attribute to a mapping, *SplitAndMerge* checks whether the mapping already contains another attribute from the same source. Clearly, if two attributes from the same source can match a mapping, their matching scores against that mapping are greater than the matching threshold. As such a threshold value would lead to a violation of the *SplitAndMerge* constraint, the algorithm can conclude that it is too low for that mapping.

A suitable threshold should keep the two attributes well separated. Therefore the matching threshold of this mapping could be set to the value of the similarity between the two attributes. However, attributes that participated the mapping before the threshold update were included by an inappropriate threshold, as well. Therefore these attributes

need to be re-aggregated in new mappings around the two conflicting attributes. This task is done by a procedure called *Split*.

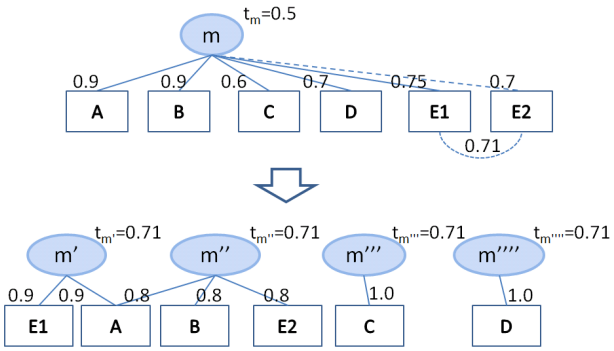


Figure 6: Application of the *Split* procedure over a mapping $m$. Labels on edges indicate matching scores and attribute similarities. Attributes $E_1$ and $E_2$ belong to the same source; $sim(E_1, E_2) = 0.71$

To illustrate the *Split* procedure, we make use of the example in Figure 6. Let $m$ be a mapping composed by attributes $\{A, B, C, D, E_1\}$, with matching threshold $t_m = 0.5$.[5] Let $E_2$ be an attribute that belongs to the same source of $E_1$. Suppose that $score(E_2, m) = 0.7$: it is greater than $t_m$, then $E_2$ should be added to $m$; but as another attribute from the same source ($E_1$) is already present in $m$, the *Split* procedure is invoked on $m$.

The *Split* procedure starts with the creation of two mappings, each initialized with one of the two attributes coming from the same source. The matching thresholds of these mappings is assigned the value of the similarity between the attributes that have triggered the procedure. Then, the initial mapping is destroyed, and it is checked whether its attributes can participate to the new mappings.

In our example, the new mappings $m' = \{E_1\}$ and $m'' = \{E_2\}$ would be created, both with matching thresholds $t_{m'} = t_{m''} = sim(E_1, E_2) = 0.71$. Assuming that $A$ matches with $m'$ and $m''$ ($score(A, m') > t_{m'}$ and $score(A, m'') > t_{m''}$), $B$ matches with $m''$ ($score(B, m'') > t_{m''}$), while $C$ and $D$ do not match with neither $m'$ nor $m''$, then $m' = \{E_1, A\}$ and $m'' = \{E_2, A, B\}$.

Attributes from the original mapping that are not included in the newly generated mappings are iteratively processed. At each step, if an attribute cannot be included in any mapping generated in the scope of the current *Split* execution, it gives rise to a new (singleton) mapping. The value of the similarity between the attributes that have triggered the procedure is assigned to the matching thresholds of all the mappings created in these steps.

In the example, $C$ originates a new mapping $m''' = \{C\}$, with $t_{m'''} = sim(E_1, E_2)$; next, if $score(D, m''') > t_{m'''}$ then $m''' = \{C, D\}$, otherwise a new singleton mapping $m'''' = \{D\}$ is created, again with $t_{m''''} = sim(E_1, E_2)$.

Note that the effects of the splitting propagate for all the remaining iterations; in particular, the similarity between the attributes that trigger the split is assigned to the matching thresholds of all the mappings generated by the procedure. This strategy can result too aggressive, because if the split has been caused by a local (possibly erroneous) property of one source, increasing the matching threshold could prevent attributes from the remaining relations to be aggregated into homogeneous mappings.

To overcome this drawback, *SplitAndMerge* adopts a more conservative strategy: whenever two attributes $A$ and $B$ match the same mapping $m$, the algorithm invokes a *Split*. However, the matching threshold is not set to $sim(A, B)$, but it is incremented by a value $\delta$, which is smaller than the difference between $sim(A, B)$ and $t_m$. In this way, the *Split* rearranges the mapping of the involved attributes with a more prudent approach. Observe that the greater is $\delta$, the less sources are needed to agree for triggering a *Split*. If $\delta = 0$ the algorithm corresponds to the naive matching algorithm; if $\delta = sim(A, B)$, the algorithm splits the mapping at the first violation of the *SplitAndMerge* constraint. To govern the splitting strategy we set $\delta = \frac{sim(A,B)-t_m}{h}$, where $h$ is an integer constant. In our setting $h = 3$: it approximatively corresponds to the requirement that at least three sources must agree before splitting.

## 4.2 Attribute Similarity and Matching Score

The similarity between two attributes $A$ and $B$ is computed by comparing the values that they assume in a small sample set of *aligned* tuple pairs. Two tuples can be aligned if they refer to the same real world object. In our framework, the

[5] Clearly, $A$, $B$, $C$, $D$ and $E_1$ belong to distinct sources.

task of identifying such a small set of tuple pairs is simplified by the results returned by the companion crawler, which associates an identifier (e.g. the company name in the stock quote example) with the collected pages. We therefore align tuples that are extracted from pages having equal identifiers. However, if these identifiers were not available, record linkage techniques for opaque relations (such as those described in [5]) could be profitably applied to this end.

Let $A$ and $B$ be attributes of the relations $r_1$ and $r_2$, respectively; we denote $t_1[j](A)$ and $t_2[j](B)$ the values assumed by $A$ and $B$ in the tuples of $r_1$ and $r_2$ associated with the real world object $j$. Given a set of $l$ aligned tuples from $r_1$ and $r_2$, the similarity between $A$ and $B$, $sim(A, B)$ is computed as follows:

$$sim(A, B) = 1 - \frac{1}{h} \sum_{j=1}^{l} f(t_1[j](A), t_2[j](B))$$

where $h$ is the number of tuples such that both $t_1[j](A)$ and $t_2[j](B)$ are not null; $f(\cdot, \cdot)$ is a pairwise similarity function that returns 0 whenever either $t_1[j](A)$ or $t_2[j](B)$ is null; $f(\cdot, \cdot)$ is a type dependant metrics: for instance, in case of numbers, it returns the normalized distance; in case of strings, it uses a standard string metrics such as the Jensen-Shannon distance.[6]

Attributes are associated with a type in a preliminary step, right after they are extracted from pages. Types are taken from a simple hierarchy including basic data types, such as *String*, *Integer*, *Double*, and high level data types such as *Date*, *Length*, *Weight*, *Currency*. The type is inferred by means of a simple parser that analyzes the syntax of the extracted values looking up a set of predefined patterns associated with every type, such as \d+ [-|/] \d+ [-|/] \d+ for *Date*. *String* is chosen when no other type applies.

At the end of this preliminary analysis, each attribute $A$ is assigned a type. For types whose values can be expressed according to different formats and/or units of measure (e.g. meters, centimeters, foots, inches, etc. for length) we also normalize the values to a reference format and unit of measure (e.g. *Length* values are expressed in centimeters).

The matching score of an attribute $A$ against a mapping $m$ is computed as follows. Let $\{B_1, B_2, \ldots, B_n\}$ be the subset of the attributes in $m$ whose source relations have at least $\sigma$ tuples that align with tuples of the relation of $A$, then:

$$score(A, m) = \frac{1}{n} \sum_{i=1}^{n} sim(A, B_i)$$

Observe that when computing the score, instances from different sources are involved; however only sources that overlap for at least $\sigma$ instances with the current relation are considered.

### 4.3 Implementation Issues

The type characterization for the extracted attributes allows the implementation of simple yet effective optimizations during the mapping inference process: many candidate matchings can be easily discarded if the types of the involved attributes do not agree. To further reduce the number of candidate matching, we also consider simple statistical features, such as average and variance for numerical attributes, string distribution, average and variance of string lengths for textual attributes. Attributes with different features cannot be matched.

Another important aspect of the implementation of the algorithm is the order in which the relations are processed. As our approach is based on the alignment of tuples, we aim at finding a processing order that at each iteration maximizes the number of overlapping tuples. To this end, each iteration picks out the relation with the maximum cardinality. If the chosen relation has not sufficient tuples that align with those already processed, it is queued and it will be processed later, hoping that in the later iterations other relations will contain tuples aligning with them.

## 5 Wrapper Refinement

The wrapper refinement process is activated at the conclusion of each iteration of the mapping inference algorithm, and it focuses on the rules of the wrapper associated with the current relation. We now describe how our algorithm leverages on the results produced during the mapping inference to refine the extraction rules in the bootstrapping wrappers.

The process involves rules corresponding to both attributes that were added to some mappings and attributes that partially matched with some mappings (weak rules). To this end the process elaborates the rules of any attribute $A$ whose matching score against a mapping $m$ was greater than a threshold $T_f$, with $T_f < t_m$.

[6] We use a variant of this metrics, namely the one provided with the library discussed in [14].

As described in the previous section, the matching score between an attribute and a mapping is computed by comparing pairwise values for a set of aligned pages. Therefore, the matching score depends on the extraction rule associated with the attribute: if the extraction rule does not work correctly for all the pages of the source, some of the extracted values might not match with those of the attributes in the mapping, thus producing a low matching score.

To give an example, consider again the extraction rule $er_{A3}$ in Figure 4: as shown in Figure 5, the application of this rule over the pages in Figure 3 returns wrong data. In particular, for some pages it extracts the minimum value of a stock, for others (those that do not publish such information) the maximum value. Suppose that the mapping $m$ in Figure 7 is present at the iteration that involves the processing of the relation $A_3$ belongs to. Because of the heterogeneous values extracted by $er_{A3}$, $A_3$ cannot match with $m$: some of its values match with those of $X$, $Y$, $Z$, while others differ, negatively contributing to $score(A_3, m)$. However, assuming that $score(A_3, m) > T_f$, the rule $er_{A3}$ will be processed in order to find a substitutive expression. The goal is to improve the matching score by extracting values that better match with the values of the attributes in $m$.

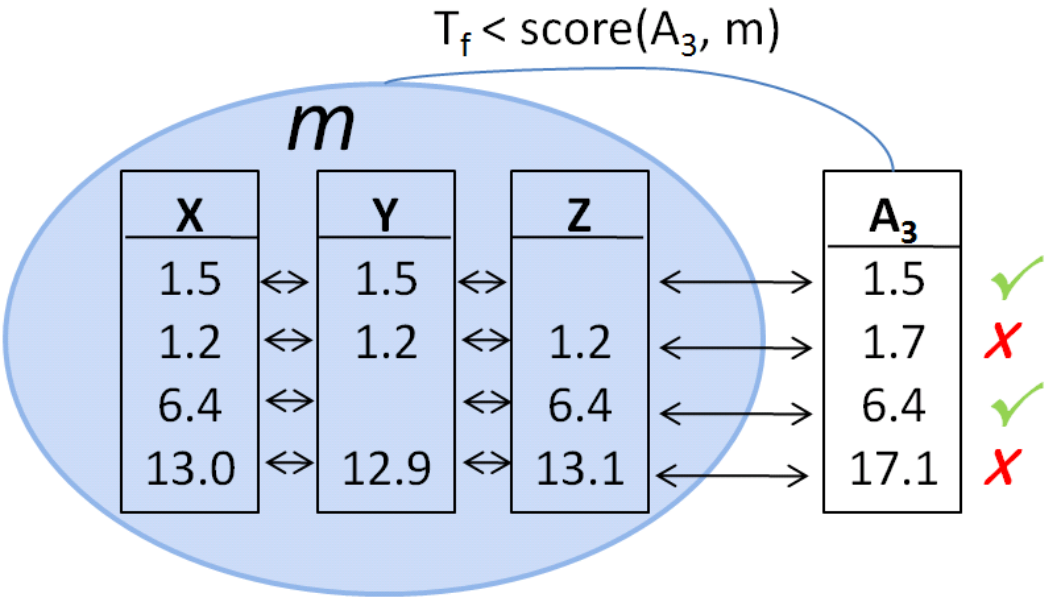


Figure 7: The values of attribute $A_3$ partially match with the values of the attributes in $m$.

We try to create candidate XPath rules to substitute a rule by leveraging on the values extracted from the other redundant sources: the alternative rule must preserve the values matching with those extracted from the other sources, while it should extract new values substituting the mismatching ones. To this end, we enumerate a set of relative XPath expressions obtained by juxtaposing two subexpressions: one locating an invariant node close to the matching values to extract; and the other reaching the values starting from the invariant's position.

For efficiency reasons, the enumeration of the candidate rules involves only a subset of the values to extract and only a subset of the closest invariant nodes to them. The former are selected by considering the matching values $c_1, \ldots, c_m$ such that every $c_i$ satisfies the following predicate:

$$\frac{1}{n} \sum_{j=1}^{n} f(c_i, q_j) \geq t_m$$

where $q_i \in Q$ is the i-th value from the set $Q$ of the associated values in the mapping and $f()$ is the pairwise similarity function presented in Section 4.2. An invariant node is considered only if it always occurs at the same distance from the value according to a simple distance metrics between DOM nodes.[7]

In the above example, the absolute $er_{A3}$ can be replaced with a relative rule based on the invariant "`min`" which is the closest invariant (occurring always at distance 0 from the two matching values `1.5` and `6.4`). The erroneously extracted values, 1.7 and 17.1, are not considered since they do not match with the other sources. For example, 17.1 is

[7] In our implementation the distance metrics measures the number of leaf nodes separating the invariant to the value node in the DOM tree.

not considered because $\frac{1}{3}[f(17.1, 13.0) + f(17.1, 12.9) + f(17.1, 13.1)] < t_m$. The obtained rule is then:
$er_{A3} \leftarrow$ //td[contains(text(),'min')]/../tr/td[2].

Since an attribute $A$ can have multiple uncertain matches with several other mappings $m_1, \ldots, m_n$, the procedure is repeated for each mapping, and an extraction rule can potentially originate several new rules. These rules are applied and then evaluated by computing again the matching scores. Those that do not improve the score are immediately discarded, according to a monotone procedure that is repeated until the matching score of the values extracted by a rule cannot be further improved. For each mapping $m_i$ if the score of the best candidate rule $r$ is both greather than the matching threshold of the mapping $t_{m_i}$, and greater than $score(A, m_i)$, then $r$ will generate a new attribute $A_r$ which replaces $A$ in $m_i$.

An important observation is that the invariant node selected by the refinement procedure quite often represent a meaningful semantics label for the associated values. This is reasonable, because these labels are part of the page template, and they are usually close to the values they describe. Therefore, we associate each mapping with all the labels occurring in the corresponding extraction rules and we rank them according to this simple formula: $score(l) = \frac{n_l}{1+d_l}$, where $n_l$ is the number of extraction rules involving the label $l$, and $d_l$ is the sum of the distances between the invariant node and the value node in those rules.

# 6 Experiments

We conducted experimental studies to evaluate the performance of our techniques. We run experiment on both synthetic and real-world scenarios to validate and compare the various techniques proposed in the paper.

## 6.1 Metrics

To evaluate the results produced by the system we propose several metrics for different purposes.

First, we are interested in evaluating the quality of the mappings automatically produced by our techniques. We use the standard metrics precision ($P$), recall ($R$), and F-measure ($F$) to evaluate each mapping $A$ generated by our algorithm with respect to a golden mapping $B$:

$$P = \frac{|A \cap B|}{|A|}; R = \frac{|A \cap B|}{|B|}; F = \frac{2 * P * R}{P + R}$$

Another metrics that we introduce to evaluate the mappings is the *cohesion* of the values of the attributes of each mapping. Given a mapping $m = \{A_1, A_2, \ldots, A_n\}$ the cohesion of $m$ is the average matching score of each attribute against the mapping:

$$C = \frac{1}{n} \sum_{i=1}^{n} score(A_i, m)$$

It is worth saying that the cohesion is strictly related to the precision defined above. However, the cohesion is computed only considering the values of the attributes involved in the mappings and thus it does not require the availability of a golden schema. To give a global evaluation of all the mapping we also provide the *global mapping cohesion*, which corresponds to the average cohesion computed over all the mappings.

**Experimental setup** For our experiments we set the matching threshold $T = 0.5$, the matching score that trigger the wrapper refinement $T_f = T/2$; the minimum number of overlapping tuples to compute their score $\sigma = 5$; and number of sources needed to arise a split event $h = 3$.

## 6.2 Experimental Results on the Synthetic Scenarios

The goal of this experiment was to evaluate the effectiveness of the dynamic matching threshold of the *SplitAndMerge* approach. To this end, we have developed a tool to automatically generate synthetic sources. The tool generates the desired number of sources, taking as input the list of attributes exposed by the sources, and the average percentage of overlapping between instances of distinct sources. In particular, for each attribute, we specify: *(i)* the type (double, string, date), *(ii)* the range of values (for strings, a vocabulary), *(iii)* the percentage of null values, *(iv)* the distribution of errors in the values. The tool first automatically generates an HTML template for each source, and then it creates pages by filling the templates with values according to the corresponding type features.

In our experiments we generated 1,000 sources, each composed by 50 pages, with 15 attributes; namely, 3 strings (random words from a vocabulary) and 12 doubles (all with different distributions); we set up a random error of 5% between the values of the same attributes across different sites to simulate the discrepancy introduced by publishers in real web sites;[8] 3 attributes have 50% of null values, to simulate the presence of optional patterns in the pages.

We ran several executions of the *NaiveMatch* and the *SplitAndMerge* algorithms: in each execution the initial value of the dynamic threshold used by the latter was set to coincide with the fixed threshold used by the former. For each execution we computed the F-measure of the generated mappings. In this experiment the set of golden mappings was known a priori, as the sources were generated by the tool.

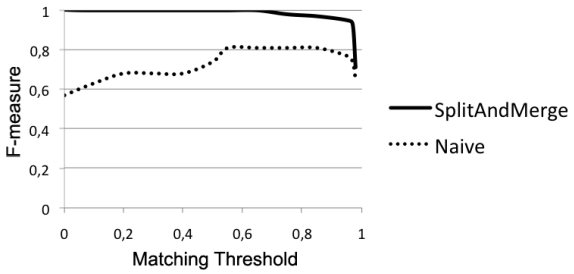


Figure 8: Comparison of the *NaiveMatch* and the *SplitAndMerge* algorithms with different thresholds.

Figure 8 reports in a graph the results of the experiment. In particular it draws the average F-measure computed over the set of output mappings. Observe that, if the starting value of the threshold is below 0.65, the *SplitAndMerge* algorithm can always dynamically improve the threshold and it reaches perfect results. Around the same threshold value, the *NaiveMatch* algorithm starts to perform well, since it reaches a good compromise between precision and recall. When the threshold reaches the value of 0.9, the F-measure for both the approaches quickly decreases due to the degradation of the recall: very high values for the matching threshold are not able to handle the discrepancies in the values.

## 6.3 Experimental Results on the Real World Scenarios

To experiment our techniques over real world scenarios, we collected several data sources from the Web over three application domains: *Soccer*, *Videogames*, and *Finance*. The sources were gathered automatically by our companion crawler [6]. For each domain, we let the crawler collect about 100 sources. Each source consists of tens to hundreds of pages, and each page contains detailed data about one instance of the corresponding domain entity (soccer player, videogame, stock quote). As expected, within the same domain, many instances are shared by several sources. The overlap is almost total for the stock quotes, while it is more articulated for the soccer players and for videogames because these domains include both large popular sites and small ones. Anyway, in all the domains each source shares a significant number of instances with at least another source. We have estimated that each soccer player instance appears on average in 1.6 sources, each videogame in 24.5 sources, and each stock quote in 92.8 sources.[9]

In the finance domain most of the attributes are double, and several attributes have very similar values (min, max, average, open, close values of a stock). The soccer domain is interesting because it includes attributes with data types that present heterogeneous formats in the various sources; for example, height and weight of players are expressed in several different units of measure (e.g. meters vs. foots and inches) and are published according to different formats (e.g. *mt1.82* vs. *182cm*). Finally, in the videogame domain most of the attributes are strings, and with respect to the other domains, the page structures are quite irregular.

**Quality of Mappings** To evaluate the quality of the mappings, for each domain we selected 20 web sources (the largest ones) and we manually built a golden set of mappings by inspecting 10 pages per source; only the attributes that were present in at least 3 sources were included in the mappings. The golden schema of the stock quote entity contains 29 attributes; those of soccer players and videogames 14 and 11, respectively.

For each domain we run four executions with the goal of evaluating the impact of the *SplitAndMerge* and of the wrapper refinement on the quality of the inferred mappings. The first execution applied only the *NaiveMatch* algo-

[8]The presence of errors in the strings values has been simulated by inserting random characters.

[9]Popular soccer players and popular videogames are present in a large number of sources; almost all the sources publish NYSE and NASDAQ stock quotes.

rithm, without any wrapper refinement; the second run again *NaiveMatch*, but it was followed by the wrapper refinement. Similarly, the third execution runs *SplitAndMerge* without the wrapper refinement, while the fourth execution run with both *SplitAndMerge* and the wrapper refinement.

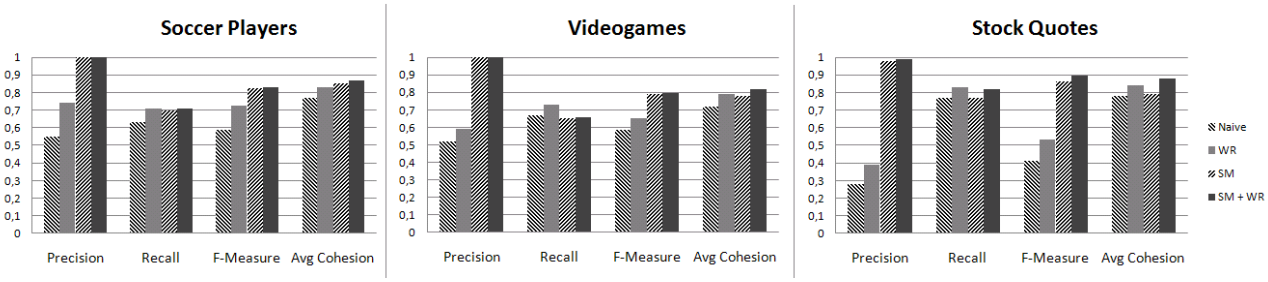


Figure 9: Precision, recall, F-measure, and global cohesion of the mappings for the four different executions: naive matching, naive matching with wrapper refinement (WR), SplitAndMerge (SM), SplitAndMerge with wrapper refinement (SM+WR).

Figure 9 reports precision, recall F-measure, and global cohesion of these experiments. It is apparent that the best performances in terms of precision and F-measure are always obtained when the approach run with both the *SplitAndMerge* and the wrapper refinement activated. In only one case, *Videogames*, it is overcome by another execution in terms of recall.

A few observations are worth here. First, *NaiveMatch* alone always obtains mappings with high recall but with low precision, especially in the finance domain. In fact, *NaiveMatch* is able to gather an high number of valid attributes, but it aggregates several heterogeneous attributes within the same mapping, as it is not able to distinguish attributes with similar values, and thus it produces many false positives.

The precision of the *SplitAndMerge* algorithm greatly benefits of the more advanced matching technique, especially in the finance domain. On the other hand, very high threshold values slightly degrade the recall results, since the erroneous data published by some sources introduce discrepancies in the values and prevent some matches.

It is interesting to observe the direct correlation between the thresholds that have been dynamically increased and the improvements in the results. The correlation is highlighted comparing the *NaiveMatch* and the *SplitAndMerge* executions in Figure 10. In the *finance* domain, which contains several similar values, the improvement of the precision is 250%.

| Domain | Threshold increment | Precision gain |
|---|---|---|
| SOCCER | 32% | 81% |
| VIDEOGAMES | 23% | 92% |
| FINANCE | 37% | 250% |

Figure 10: Effect of the dynamic matching threshold on the mapping Precision.

The wrapper refinement has always positive impacts on the performance. First, it increases both precision and recall; as extraction rules are improved some attributes can reach a sufficient matching score to be added in the mappings set. Second, it significantly improves the global coherence: this is a clear consequence of the improved quality of the wrappers.

To study the influence of redundancy of data on the performance of our techniques we have computed precision, recall, F-measure and global cohesion considering only mappings that involve attributes that appear more frequently; in particular we computed the values of our evaluation metrics for mappings referring to attributes that appears in at least 8 sources. Figure 11 reports the results of this setting. Overall, we observe an improvement of the F-measure, which is mainly due to a higher recall. Interestingly, for these mappings the wrapper refinement has a strong influence on the precision. This means that the presence of redundant information can contribute to improve the wrappers.

**Scalability** To evaluate the scalability of the approach, we have run the system against the whole sets of data sources. Because of the large number of sources, it was unfeasible to manually create the golden mappings. Therefore, to give a quantitative evaluation of the results, we report detailed data about a number of mappings obtained by running the

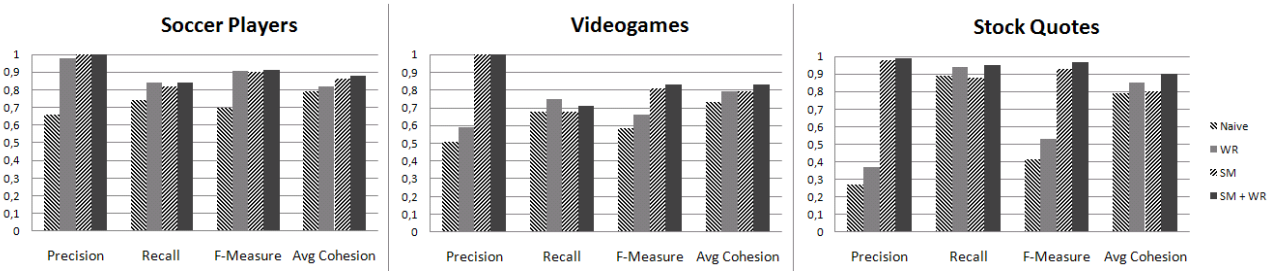


Figure 11: Precision, recall, F-measure, and global cohesion for mappings composed by attributes that appeared in at least 8 sources.

system with both *SplitAndMerge* and the wrapper refinement. In Figure 12 we illustrate, for the 8 largest mappings: *(i)* the cardinality of the mapping (|m|), i.e. number of distinct attributes; *(ii)* the cohesion (C); *(iii)* the first two labels assigned by the system; for each label, we also report the value of the score normalized with respect to the highest value.

| SOCCER PLAYERS 45,714 PAGES | | | | VIDEOGAMES 49,262 PAGES | | | | STOCK QUOTES 57,623 PAGES | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Attribute** | **\|m\|** | **C** | **Labels (score)** | **Attribute** | **\|m\|** | **C** | **Labels (score)** | **Attribute** | **\|m\|** | **C** | **Labels (score)** |
| Name | 90 | 0.85 | Name (1.0)<br>Nationality (0.2) | Title | 86 | 0.77 | Publisher (1.0)<br>Title (0.76) | Stock Symbol | 84 | 0.80 | Industry (1.0)<br>Stock (0.30) |
| Birth Date | 61 | 0.91 | Birth (1.0)<br>Date (0.7) | Publisher | 59 | 0.72 | Publisher (1.0)<br>Release (0.14) | Price Change | 73 | 0.89 | Price (1.0)<br>Trade (0.92) |
| Height | 54 | 0.99 | Height (1.0)<br>Weight (0.1) | Developer | 45 | 0.71 | Developer (1.0)<br>Genre (0.14) | % Change | 73 | 0.90 | Change (1.0)<br>Bid (0.90) |
| Nationality | 48 | 0.84 | Nationality (1.0)<br>Born (0.4) | Genre | 28 | 0.76 | Genre (1.0)<br>Release (0.53) | Volume | 52 | 0.92 | Volume (1.0)<br>Vol (0.23) |
| Club | 43 | 0.71 | Club (1.0)<br>History (0.5) | Esrb Rating | 20 | 0.79 | Rating (1.0)<br>Esrb (0.61) | Day Low | 43 | 0.98 | Low (1.0)<br>Beta (0.7) |
| Position | 43 | 0.69 | Position (1.0)<br>Clubs (0.1) | Release Date | 9 | 0.80 | Release (1.0)<br>Date (0.91) | Day High | 41 | 0.99 | High (1.0)<br>Previous (0.56) |
| Weight | 34 | 0.96 | Weight (1.0)<br>Height (0.1) | Platform | 9 | 0.75 | Platform (1.0)<br>Developer (0.12) | Last Price | 29 | 0.99 | Last (1.0)<br>Price (0.38) |
| League | 14 | 0.66 | 2005 (1.0)<br>Uefa (0.8) | # Players | 6 | 0.81 | Players (1.0)<br>Esrb (0.11) | Open Price | 24 | 0.99 | Open (1.0)<br>Previous (0.23) |

Figure 12: Top-8 results for 100 web sources. For each mapping $m$ the table reports the attribute name (manually assigned), the mapping cardinality, the mapping cohesion, and the top-2 automatically assigned labels with their normalized scores.

The system correctly assigns labels to 21 out of 24 attributes. Observe that for most of the labels the score difference between the first and the second choice is rather high, while it is low for two of the wrong labels (LEAGUE in the *Soccer* domain, TITLE in *Videogames*).

The cohesion of mappings is higher for the stock quote attributes. We know that in this domain many attributes have very similar values; the high cohesion in the values of these attributes suggests they have been correctly extracted.

# 7 Related Work

Extraction and integration of data from the Web is a challenging issue and some proposals have recently attacked the problem at the web scale.

**Information Extraction** DIPRE [9] represents a pioneering technique to extract relations from the Web. Starting from a bunch of seed pairs (e.g. some author-book pairs), DIPRE collects thousands of similar pairs by means of a process in which the research of new pages containing these pairs and the inference of patterns extracting them are interleaved. The applicability of the approach is limited, since it cannot deal with generic tuples of more than two

attributes, but the paper motivated several web information extraction projects to develop effective techniques for the extraction of facts from large corpora of web pages [2, 18, 4]. Facts are collections of named-entities (such as, for example, names of scientists, politicians, cities), and binary predicates, e.g. born-in⟨politician, city⟩. Web information extraction techniques mainly infer lexico-syntactic patterns from textual descriptions, but they cannot take advantage of the structures available on the Web, as they fail to elaborate data that are embedded in HTML templates, such as those typically published in data intensive web sites. In a data integration perspective, these information extraction methods are able to automatically produce semantic mappings for a huge amount of data extracted from the Web. These solutions scale on the extension, as they produce huge relations from web data, but not in the intension (schema), as the produced relations express at most binary associations.

**Data Integration for Web Data** The problem of designing effective data integration solutions has been addressed by several research and development projects in the last years, but data integration is still a challenging issue [19, 8]. In our system schematic and identity conflicts are handled with the matching and the record linkage steps, respectively. Data conflicts (i.e., distinct sources reporting alternative values for an attribute of the same object) are tackled with a conflict resolution strategy based on a consensus model, but its description is out of the scope of this paper [7]. A data integration architecture for structured data on the Web is the subject of the PAYGO project [22]. However PAYGO focuses on explicitly structured sources, such as Google Base, while our approach aims at integrating data from web sites. An automatic technique to bootstrap data integration systems has been recently presented [25], but it does not consider the data extraction problem, while the focus of our work is the interaction between integration and extraction processes. Also, the proposed integration techniques are based on the availability of attribute labels; on the contrary, our system deals with unlabeled data and automatically infers the labels whenever they are encoded in the HTML templates.

The exploitation of structured data on the Web is the primary goal of WebTables [11], which concentrates on the vast amount of data which are published in HTML tables. Compared to information extraction approaches, WebTables extracts relations with involved relational schemas. However, it does not address the issue of integrating the extracted data. We observe that the data extracted by WebTables can be hardly integrated because of the heterogeneity of the sources, and because data are scattered in a huge number of (small) relations. Also, the wrapper generation process in WebTables is simplified because the approach concentrates only on tabular data published in HTML tables. Cafarella et al. have described a system to populate a probabilistic database with data extracted from the Web [10]. However data extraction is performed by TextRunner [4], an information extraction system that suffers the problems of working on data rich web pages discussed above. The goal of the CIMPLE project is the development of a platform to support the information needs of the members of a virtual community [26, 16]. Compared to our method, CIMPLE requires an expert to provide a set of relevant sources, to design an E-R schema describing the domain of interest, and to compose the operators for the extraction of the data from the pages. MetaQuerier also aims at enabling effective access to structured information sources on the Web [12]. One of the main issues that have been addressed in this project deals with the automatic definition of schema matching across web query interfaces (forms) [20].

**Schema matching** A challenging problem in our context is automatic schema matching [24]. The idea of using duplicate instances in the matching process has been recently studied [5, 27]: these proposals show how the redundancy can help in contexts where schema can be imprecise. However they do not face the scalability issues that arise when dealing with web data. Data instances and domain constraints are used also in Glue [17], which early introduced a framework for finding semantic mappings between concepts of ontologies. Although the Glue approach has a general applicability in the semantic web context, it is not suitable in our setting, since it relies also on elements names of the ontology taxonomy and on the hierarchical relationships among elements. Our wrapper refinement phase resembles the intuitions behind the "augment method" in [21], with the remarkable difference that we automatically gather the corpus during the integration process while in their case the corpus is given as input. In fact, a direct application of their approach is not possible in our setting, since we do not consider a-priori information about the domain (i.e., at bootstrap we do not have any corpus of schemas nor mappings).

**Wrapper Inference** An important issue in our solution is the automatic generation of wrappers. RoadRunner [15] and ExAlg [3] propose automatic inference techniques to create a wrapper from a collection of pages generated by the same template. However these approaches are not suitable to our goals, because they do not scale with the number of sources, and because they generate complex nested schemas that can be hard to integrate when dealing with a large number of sources.

A more closely related approach is developed in the TurboWrapper project [13]: it introduces a complex architecture that includes several wrapper inference systems (e.g. RoadRunner and ExAlg themselves) with the goal of improving the results of the single partecipating system taken separately. Interestingly, the design of TurboWrapper is motivated by the observation that different web sources that offer data for the same domain have a strong redundancy

at the schema level. However they do not consider the redundancy of information that occurs at the instance level, and the correlation of data in the integration step is based on the assumptions that there exists a generative model for some attributes (e.g. the *isbn* in the book domain). This is a strong domain dependent assumption that limits the application of the technique, since in several domains, such as the finance domain, distinct attributes might have a too similar underlying generative model, e.g. min and max of a stock quote.

# 8 Conclusions

We showed that it is possible to automatically obtain high quality mediated schemas and mappings for redundant and data-intensive web sites. This process relies on the great abundance of overlapping data over the Web and exploits such redundancies without any domain specific technique and employing only non supervisioned solutions. Our techniques built on modeling such redundancies and adapt well to the discrepancies that occur in such settings. The main novel elements we introduced are the *SplitAndMerge* algorithm and the refinement procedure. We showed that exploiting the opportunities given by a large amount of sources to integrate, it is possible to achieve better precision and recall in the matching of the schemas. We think that the refinement technique brings a contribution for web settings, when there is the need to automatically generate wrappers: relying on the redundancies of already integrated data, we showed that it is possible to significantly improve the level of automation of the state-of-the-art wrapper generator systems. There are many interesting directions for developing our approach: in this paper we focused on schema made of just one relation, but an interesting opportunity is how to deal with entire data intensive web sites, with multiple entities and relationships involved.

# References


[1] The claremont report on database research. *SIGMOD Rec.*, 37(3):9–19, 2008.

[2] Eugene Agichtein and Luis Gravano. Snowball: extracting relations from large plain-text collections. In *DL '00*, pages 85–94, 2000.

[3] A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *ACM SIGMOD International Conf. on Management of Data (SIGMOD'2003), San Diego, California*, pages 337–348, 2003.

[4] M. Banko, M. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the web. In *IJCAI*, 2007.

[5] Alexander Bilke and Felix Naumann. Schema matching using duplicates. In *ICDE*, pages 69–80, 2005.

[6] Lorenzo Blanco, Valter Crescenzi, Paolo Merialdo, and Paolo Papotti. Supporting the automatic construction of entity aware search engines. In *WIDM*, pages 149–156, 2008.

[7] Lorenzo Blanco, Valter Crescenzi, Paolo Merialdo, and Paolo Papotti. A probabilistic model to characterize the uncertainty of web data integration: What sources have the good data? Technical report, DIA - Roma Tre - TR146, June 2009.

[8] Jens Bleiholder and Felix Naumann. Data fusion. *ACM Comput. Surv.*, 41(1), 2008.

[9] S. Brin. Extracting patterns and relations from the World Wide Web. In *Proceedings of the First Workshop on the Web and Databases (WebDB'98) (in conjunction with EDBT'98)*, pages 102–108, 1998.

[10] Michael J. Cafarella, Oren Etzioni, and Dan Suciu. Structured queries over web text. *IEEE Data Eng. Bull.*, 29(4):45–51, 2006.

[11] Michael J. Cafarella, Alon Y. Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. Webtables: exploring the power of tables on the web. *PVLDB*, 1(1):538–549, 2008.

[12] Kevin Chen-Chuan Chang, He Bin, and Zhang Zhen. Toward large scale integration: Building a metaquerier over databases on the web. In *CIDR 2005*, pages 44–66, 2005.

[13] Shui-Lung Chuang, Kevin Chen-Chuan Chang, and Cheng Xiang Zhai. Context-aware wrapping: Synchronized data extraction. In *VLDB*, pages 699–710, 2007.

[14] William W. Cohen, Pradeep Ravikumar, and Stephen E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWeb*, pages 73–78, 2003.

[15] V. Crescenzi, G. Mecca, and P. Merialdo. ROADRUNNER: Towards automatic data extraction from large Web sites. In *International Conf. on Very Large Data Bases (VLDB 2001), Roma, Italy, September 11-14*, pages 109–118, 2001.

[16] Pedro DeRose, Warren Shen, Fei Chen, AnHai Doan, and Raghu Ramakrishnan. Building structured web community portals: A top-down, compositional, and incremental approach. In *VLDB*, pages 399–410, 2007.

[17] AnHai Doan, Jayant Madhavan, Pedro Domingos, and Alon Halevy. Learning to map between ontologies on the semantic web. In *WWW '02*, pages 662–673, 2002.

[18] Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S.Weld, and Alexander Yates. Unsupervised named-entity extraction from the web: An experimental study. *Artificial Intelligence*, 165:91–134, 2005.

[19] Laura M. Haas. Beauty and the beast: The theory and practice of information integration. In *ICDT*, pages 28–43, 2007.

[20] Bin He and Kevin Chen-Chuan Chang. Automatic complex schema matching across web query interfaces: A correlation mining approach. *ACM Trans. Database Syst.*, 31(1):346–395, 2006.

[21] Jayant Madhavan, Philip A. Bernstein, AnHai Doan, and Alon Y. Halevy. Corpus-based schema matching. In *ICDE*, pages 57–68, 2005.

[22] Jayant Madhavan, Shirley Cohen, Xin Luna Dong, Alon Y. Halevy, Shawn R. Jeffery, David Ko, and Cong Yu. Web-scale data integration: You can afford to pay as you go. In *CIDR 2007*, pages 342–350, 2007.

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

[24] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.

[25] Anish Das Sarma, Xin Dong, and Alon Y. Halevy. Bootstrapping pay-as-you-go data integration systems. In *SIGMOD Conference*, pages 861–874, 2008.

[26] Warren Shen, Pedro DeRose, Robert McCann, AnHai Doan, and Raghu Ramakrishnan. Toward best-effort information extraction. In *SIGMOD Conference*, pages 1031–1042, 2008.

[27] Xuan Zhou, Julien Gaugaz, Wolf-Tilo Balke, and Wolfgang Nejdl. Query relaxation using malleable schemas. In *SIGMOD Conference*, pages 545–556, 2007.