Redundancy-Driven Web Data Extraction and Integration

Lorenzo Blanco, Mirko Bronzi, Valter Crescenzi, Paolo Merialdo, Paolo Papotti
Università degli Studi Roma Tre
Dipartimento di Informatica e Automazione
{blanco,bronzi,crescenz,merialdo,papotti}@dia.uniroma3.it

ABSTRACT

A large number of web sites publish pages containing structured information about recognizable concepts, but these data are only partially used by current applications. Although such information is spread across a myriad of sources, the web scale implies a relevant redundancy. We present a domain independent system that exploits the redundancy of information to automatically extract and integrate data from the Web. Our solution concentrates on sources that provide structured data about multiple instances from the same conceptual domain, e.g., financial data, product information. Our proposal is based on an original approach that exploits the mutual dependency between the data extraction and the data integration tasks. Experiments on a sample of 175,000 pages confirm the feasibility and quality of the approach.

1. INTRODUCTION

An increasing number of web sites deliver pages containing structured information about recognizable concepts, relevant to specific application domains, such as stock quotes, athletes, movies. For example, the pages shown in Figure 1 contain structured information about stock quotes. Current search engines are limited in exploiting the data offered by these sources. In fact the development of scalable techniques to *extract* and *integrate* data from fairly structured large corpora available on the Web is a challenging issue, because to face the web scale these activities should be accomplished automatically by domain independent techniques.

To cope with the complexity and the heterogeneity of web data, state-of-the-art approaches focus on information organized according to specific patterns that frequently occur on the Web. Meaningful examples are presented in [8], which focuses on data published in HTML tables, and in [2], which exploits lexical-syntactic patterns. As noticed in [8], even if a small fraction of the Web is organized according to these patterns, due to the web scale the amount of data involved is impressive.

WebDB '10 Indianapolis, IN USA

In large data intensive web sites, we observe two important characteristics that suggest new opportunities for the automatic extraction and integration of web data. On the one hand, we observe *local regularities*: in these sites, large amounts of data are usually offered by hundreds of pages, each encoding one tuple in a local HTML template. For example, pages shown in Figure 1 (which are from three different sources) publish information about one company stock. If we abstract this representation, we may say that each web page displays a tuple, and that a collection of pages provided by the same site corresponds to a relation. According to this abstraction, the web sites for pages in Figure 1 expose their own "StockQuote" relation. On the other hand, we notice *global information redundancy*: as the Web scales, many sources provide similar information. The redundancy occurs both a the schema level (the same attributes are published by more than one source) and at the extensional level (some objects are published by more than one source). In our example, many attributes are present in all the sources (e.g., the company name, last trade price, volume); while others are published by a subset of the sources (e.g., the "Beta" indicator). At the extensional level, there is a set of stock quotes that are published by more sources. As web information is inherently imprecise, redundancy also implies inconsistencies; that is, sources can provide conflicting information for the same object (e.g., a different value for the volume of a given stock).

These observations lead us to hypothesize that underlying sources of the same domain there is a *hidden conceptual relation*, with schema $\mathcal{R}(A_1, \ldots, A_n)$, from which pages of different sources are generated. According to this model, each of the sources $\{S_1, \ldots, S_m\}$ can be seen as the result of a generative process applied over the hidden relation. Each source publishes information about a subset of the tuples in the hidden relation, and different sources may publish different subsets of its attributes. Moreover, the sources may introduce errors, imprecise or null values, or they may publish values by adopting different formats (e.g. miles vs. kilometers). For each source S_i, we can abstract the page generation process as the application of the following operators over the hidden relation:

- *Selection* σ_i: selects a subset of the tuples;
- *Projection* π_i: selects a subset of the conceptual attributes;
- *Error* e_i: introduces errors and null values;
- *Encode* λ_i: produces a web page for each tuple by embedding its values in an HTML template.

From this perspective, the data extraction and integration problem can be seen as the problem of inverting the

Figure 1: Three web pages containing data about stock quotes from Yahoo!, Reuters, Google.

functions λ_i by specifying a wrapper $w_i = \lambda_i^{-1}$ for each source S_i. Each wrapper is defined as a set of extraction rules $w_i = \{er_{A1}, \ldots, er_{An}\}$ such that each extraction rule er_{Aj} either describes how to extract A_j values from pages of source S_i or $er_{Aj} = \perp$ if S_i does not publish the values of the conceptual attribute A_j.

Observe that the functions λ_i^{-1} define a solution both to the data extraction problem and to the integration problem, since the wrappers lead to a set of mappings by grouping the extraction rules by the underlying conceptual attribute.

A natural solution to the problem is a waterfall approach, where a schema matching algorithm is applied over the relations returned by automatically generated wrappers. However, important issues arise when a large number of sources is involved, and a high level of automation is required:

1. Wrapper Inference Issues: since wrappers are automatically generated by an unsupervised process, they can produce imprecise extraction rules (e.g., rules that extract irrelevant information mixed with data of the domain). To obtain correct rules, the wrappers should be evaluated and refined manually.

2. Integration Issues: the relations extracted by automatically generated wrappers are "opaque", i.e., their attributes are not associated with any reliable semantic label. Therefore the matching algorithm must rely on an instance-based approach, which considers attribute values to match schemas. However, due to errors introduced by the publishing process, instance-based matching is challenging as the sources may provide conflicting values. Also, imprecise extraction rules return wrong, and thus inconsistent, data.

Our techniques to compute the functions λ_i^{-1} look for the correct extraction rules and mappings contextually, and leverage the redundancy among the sources to guide the choice of the rules and their grouping into cohesive mappings. At the end of the process, we also rely on the evidence accumulated in order to extract from the page templates suitable semantic labels for the mappings.

Summary of Contributions We introduce an automatic, domain independent approach that exploits the redundancy of data among the sources to support both the extraction and the matching steps. In a bootstrapping phase, an unsupervised wrapper inference algorithm generates a set of extraction rules for each source. A novel domain independent matching algorithm compares data returned by the generated extraction rules among different sources and infers mappings. The abundance of redundancy among web sources allows the system to acquire knowledge about the domain and triggers an evaluation of the mappings. Based on the quality of the inferred mappings, the matching process provides a feedback to the wrapper generation process, which is thus driven to refine the bootstrapping wrappers in order to correct imprecise extraction rules. Better extraction rules generate better mappings thus improving the quality of the solution.

The specific contributions of the paper are the following:

- We introduce an instance based schema matching algorithm which is adaptive to the actual data. It presents significant advantages in general settings where no a priori knowledge about the domain is given, and multiple sources have to be matched (Section 2).
- Our approach takes advantage of the coupling between the wrapper inference and the data integration tasks to improve the quality of the wrappers. To the best of our knowledge, it is the first attempt to face both issues contextually (Section 3).
- We conducted a large set of experiments using real-world data from three different domains, including 300 web sites for a total of 175,000 pages. The experiments demonstrate that our techniques are effective and highlight the impact of their components (Section 4).

Related works are discussed in Section 5.

2. SOURCE INTEGRATION

In our context, a data source S is a collection of pages from the same web site, $S = \{p_1, \ldots, p_n\}$, such that each page publishes information about one object from a domain of interest. We developed a crawling algorithm [4] to locate collections of pages that publish data according to this publishing strategy. The crawler takes as input a small set of pages for an entity of interest. Hence, it explores the Web to gather collections of pages from sites delivering data about the same entity.

We describe a wrapper w as an ordered set of extraction rules, $w = \{er_1, \ldots, er_k\}$, 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 $t = \langle er_1(p), \ldots, er_k(p)\rangle$; 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: $R(A_1, \ldots, A_k)$. In a bootstrapping phase, for each source S_i an automatic wrapper inference system (we use RoadRunner [10]) generates a wrapper w_i. As the process is unsupervised, the wrappers can contain *weak* rules, i.e., rules that do not correctly extract data for all the source pages. As we shall discuss later, weak rules are refined contextually with the integration process.

A *mapping* m is a set of attributes with the same semantics coming from distinct sources. Given the set of bootstrapping wrappers $\mathcal{W} = \{w_1, \ldots, w_m\}$ (one per source S_i) that extract a set of relations $r_1 = w_1(S_1), \ldots, r_m = w_m(S_m)$ from the sources $\mathcal{S} = S_1, \ldots, S_m$, our goal is to infer a set of mappings $\mathcal{M} = \{m_1, \ldots, m_k\}$ that correlate the attributes of the relations $r_1, \ldots, r_m$. Mappings are computed according to an *attribute similarity* metric that works at the instance level (bootstrapping wrappers produce anonymous relations).

The similarity between two attributes A and B is computed by comparing the values they assume in a small sample set of *aligned* tuple pairs. Two tuples are aligned if they refer to the same real world object. In our framework, the task of identifying such a small set of tuple pairs is simplified by the results returned by the companion crawler [4], which automatically associates an identifier (e.g. the company name in the stock quote example) with the collected pages. However, if these identifiers were not available, record linkage techniques for opaque relations (such as those described in [3]) 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. Function $f(\cdot, \cdot)$ is a type dependent metric: in case of numbers, it returns the normalized distance; in case of strings, it uses a standard string distance (e.g., Jensen-Shannon).

2.1 Naive Matching

A naive 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 denote as $score(A, m)$ the *matching score* of A against m. Let $\{B_1, \ldots, B_n\} \in m$ be the attributes whose relations have at least α tuples that align with tuples of the relation of A, then: $score(A, m) = \frac{1}{n} \sum_{i=1}^{n} sim(A, B_i)$. An attribute A can participate in a mapping m if the matching score $score(A, m)$ is greater than the matching threshold t_m.

Once the initial set of singleton mappings is created, the algorithm iterates over the other relations. The algorithm processes the relations maximizing the number of overlapping tuples: each iteration picks out the relation with the maximum cardinality. If the chosen relation does not have sufficient tuples that align with those of relations already processed, it is queued and it will be processed later. Each iteration of the algorithm computes the matching score of each attribute of the current relation against all the existing mappings. If the matching score of an attribute A against a mapping m is greater than t_m, then A is added to m. If an attribute cannot match with any of the existing mappings, it gives rise to a new singleton mapping, and its matching threshold t_m is assigned the default value T.

Observe that an attribute may not match with any mapping because of the weakness of its corresponding extraction rule. The extracted data can thus match only partially with those of an existing mapping. Then, the matching scores represent a feedback about the correctness of the wrappers: rules associated to attributes that have a low matching score against some mapping are potential weak rules; they will be corrected as discussed in the next Section. The algorithm concludes when all the relations have been processed.

This 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 of attributes with different semantics. The choice of a threshold that represents a nice trade-off between precision and recall is not trivial. To address these issues we have developed a clever matching algorithm, called *SplitAndMerge*, that dynamically computes the matching threshold for every mapping and balances precision and recall.

2.2 SplitAndMerge Matching

Algorithm *SplitAndMerge* is based on the observation that it is unlikely that a source publishes the same attribute more than once, with different values. We therefore assume that non identical attributes from the same source have always different semantics. As a consequence, we impose the *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, the algorithm checks whether the mapping already contains another attribute from the same source. Clearly, if two attributes from the same source 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 above constraint, the algorithm concludes that it is too low for that mapping. A suitable threshold that would keep the two attributes separated is the value of the similarity between the two attributes. However, attributes that participated in the mapping before the threshold update were included by an inappropriate threshold, as well: these attributes need to be re-aggregated in new mappings around the two conflicting attributes.

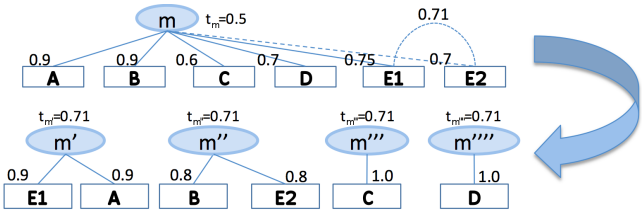

Figure 2: *SplitAndMerge* over a mapping m. Labels on edges indicate matching scores. E_1 and E_2 belong to the same source; $sim(E_1, E_2) = 0.71$.

Consider the example in Figure 2. Let m be a mapping composed by attributes $\{A, B, C, D, E_1\}$, with matching threshold $t_m = 0.5$. 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. However m already contains E_1, which comes from the same source of E_2, thus violating the constraint. In these cases, *SplitAndMerge* creates two new mappings, each initialized with one of the two attributes coming from the same source. The matching thresholds of these mappings are assigned the value of the similarity between the attributes that have trig-

gered the process. Then, the initial mapping is erased, and it is checked whether its attributes can participate in 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 both m' and m'' ($score(A, m') > t_{m'}$ and $score(A, m'') > t_{m''}$) and $score(A, m') > score(A, m'')$, B matches with m'' ($score(B, m'') > t_{m''}$), C and D do not match with neither m' nor m'', then $m' = \{E_1, A\}$ and $m'' = \{E_2, B\}$.

As a final step, attributes from the original mapping that are not included in the newly generated mappings (because their matching score would be lower than the new thresholds) are re-aggregated. If an attribute cannot be included in any mapping generated in the scope of the current 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 $t_{m'''} = sim(E_1, E_2)$; as $score(D, m''') < t_{m'''}$ a new mapping $m'''' = \{D\}$ is created, again with $t_{m''''} = sim(E_1, E_2)$.

Note that the effects of the algorithm propagate for all the remaining iterations; the similarity between the attributes that trigger the split is assigned to the matching thresholds of all the mappings generated by the procedure.

3. WRAPPER REFINEMENT

The key idea behind the wrapper refinement process is that correct rules extract consistent information from different sources; conversely, the values returned by a weak rule only partially overlap with those of other sources. Therefore, a weak rule extracts inconsistent data for some tuples, thus producing low but not negligible scores with the available mappings.

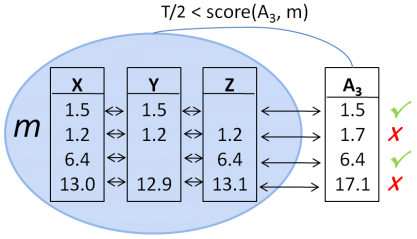

Figure 3: The values of attribute A_3 partially match with the values of the attributes in m.

The refinement process corrects the wrapper by replacing a weak rule with an alternative one that extracts consistent values, thus improving the matching score. To illustrate the technique, consider the example in Figure 3. Suppose that A_3 is matched against the mapping m, with current mapping threshold $t_m = 0.9$. Because of the heterogeneous values extracted by er_{A3}, it cannot match with m, since $score(A_3, m) = 0.87 < t_m$: some of its values (1.5 and 6.4) match with those of X, Y, Z, while others differ significantly (1.7 and 17.1), negatively contributing to the score.

Overall, the main steps of the refinement algorithm, can be summarized as follows: (i) it considers as weak all the rules of any attribute A whose matching score against a mapping m is such that $T/2 < score(A, m) < t_m$; (ii) it selects all the *matching* values v extracted by the weak rule, that is, the values that singularly contribute enough to overcome the mapping threshold; (iii) it generates an alternative rule that extracts the same matching values and maximizes the matching score of the corresponding attribute against m.

In the second step, the matching values are selected as the values v satisfying the predicate: $1 - \frac{1}{n}\sum_{j=1}^{n} f(v, q_j) \geq t_m$ where $q_1, \ldots, q_n$ are the corresponding values in the mapping and $f()$ is the pairwise similarity function. In the above example, the erroneously extracted value 17.1 is discarded since: $0.76 = 1 - \frac{1}{3}[f(17.1, 13.0) + f(17.1, 12.9) + f(17.1, 13.1)] < t_m = 0.9$.

In our implementation, the rules generated in the latter step are relative XPath expressions pivoting on a template node occurring close[1] to the matching values. More complex classes of extraction rules could be adopted. However, our experiments show that the results would be negligibly affected.

The best candidate rule to substitute a weak rule associated with an attribute A is used only if it is both greater than the matching threshold of the mapping t_m, and greater than $score(A, m)$: the new rule will generate an attribute which replaces A in m.

Adding Labels. After all the sources have been processed (i.e., wrappers have been refined and the final mappings have been computed), in a last step we determine the labels of the mappings. For each rule participating in a mapping, we process the pages of the extracted values looking for meaningful semantic labels. We leverage the observation that the labels of many attributes are part of the page template and occur close to the values.

Our technique returns as candidate labels the textual template nodes that occur closest to the extracted values. This technique may perform poorly on a single source, but it leverages the redundancy among a number of websites to select the best candidate labels. In other words, it is very unlikely that the same wrong label is associated with a mapping as frequently as a meaningful one. Therefore, we associate each mapping with all the labels associated with the corresponding rules, and then we rank the labels according to the following formula: $score(l) = \frac{n_l}{1+d_l}$, where n_l is the number of extraction rules associated to the label l, and d_l is the sum of the distances between template nodes and extracted values.

4. EXPERIMENTS

To evaluate our approach we analyzed the mappings automatically produced by our techniques. We use the standard metrics precision (P), recall (R), and F-measure (F): for each mapping A generated by our algorithm with respect to a golden mapping B we compute: P = $\frac{|A \cap B|}{|A|}$; R = $\frac{|A \cap B|}{|B|}$; F = $\frac{2*P*R}{P+R}$. For our experiments we set $T = 0.5$, $\alpha = 5$.

4.1 Results for Synthetic Scenarios

We evaluated the effectiveness of the dynamic matching thresholds of the *SplitAndMerge* algorithm against a set of synthetic sources: we generated 1,000 sources, each composed of 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

[1]The distance is measured according to a metric counting the number of leaf nodes separating the template node to the value node in the DOM tree.

sites; 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.

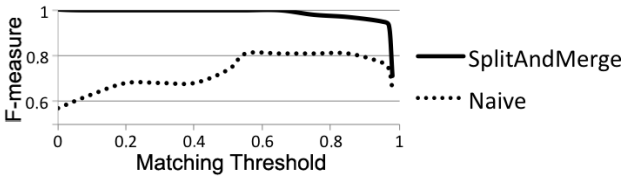

Figure 4: Comparison of the *NaiveMatch* and the *SplitAndMerge* algorithms with different thresholds.

Figure 4 reports 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 always dynamically improves the threshold reaching 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 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.

4.2 Results for Real World Scenarios

We collected several data sources from the Web over three application domains: *Soccer*, *Videogames*, and *Finance*. For each domain, we let our companion crawler [4] collect 100 sources. Each source consists of tens to thousands of pages, and each page contains detailed data about one object of the corresponding entity: Figure 5.(a) reports the increase of distinct real-world objects over the number of the sources processed. Within the same domain several objects are shared by several sources. The overlap is almost total for the stock quotes, while it is more articulated for soccer players and videogames as these domains include both large popular sites and small ones. We estimated that each soccer player object appears on average in 1.6 sources, each videogame in 24.5 sources, and each stock quote in 92.8 sources.

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 ran four executions to evaluate the impact of the *SplitAndMerge* and of the wrapper refinement on the quality of the inferred mappings. Figure 5.(b) reports precision, recall, and F-measure of the experiments. The best performances in terms of precision and F-measure are always obtained when both the *SplitAndMerge* and the wrapper refinement were activated. In only one case, *Videogames*, it is overcome by another execution in terms of recall.

A few observations are worth noting 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 many valid attributes, but it aggregates several heterogeneous attributes within the same mapping, as it is not able to distinguish attributes with similar values, thus producing many false positives. The precision of the *SplitAndMerge* algorithm greatly benefits of the more advanced matching technique, especially in the finance domain. Only in the *Videogames* domain, a very high threshold value slightly degrade the recall results, since 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 Table 1. 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%

Table 1: Effect of the dynamic matching threshold on the mapping Precision.

The wrapper refinement has always positive impacts on the performance since 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.

Soccer players 45,714 pages *(28,064 players)*		Videogames 68,900 pages *(25,608 videogames)*		Stock quotes 56,904 pages *(576 stock quotes)*	
Label	**|m|**	**Label**	**|m|**	**Label**	**|m|**
Name	90	Title	86	Stock	84
Birth	61	Publisher	59	Price	73
Height	54	Developer	45	Change	73
Nationality	48	Genre	28	Volume	52
Club	43	Rating	20	Low	43
Position	43	Release	9	High	41
Weight	34	Platform	9	Last	29
League	14	Players	6	Open	24

Table 2: Top-8 results for 100 web sources: for each mapping m the most likely label and the mapping cardinality are reported.

To give a quantitative evaluation of the results, we ran the system against the whole sets of data sources. Table 2 reports, for each of the 8 largest output mappings, the mapping cardinality $|m|$ (i.e, the number of extraction rules in each of them) and the most likely label inferred by the system. We observe that both popular attributes and rare attributes are have been correctly extracted and aggregated in the correct mapping in all the domain. Interestingly also the labels automatically associated by the system with each wrapper are correct. It is worth saying that also identification of the correct labels relies on the redundancy of information. In fact, we rely on the evidence accumulated by collecting many possible labels for the same attribute. Achieving the same precision in general is not possible considering only one site at the time: labels are not always present, are difficult to associate to the correct attribute, can be in different languages, and so on. This explains why we keep opaque the relations inferred by the wrappers until we have collected enough evidence to assign reliable labels.

Data Extension According to the model introduced in Section 1, each source provides only a partial view of the

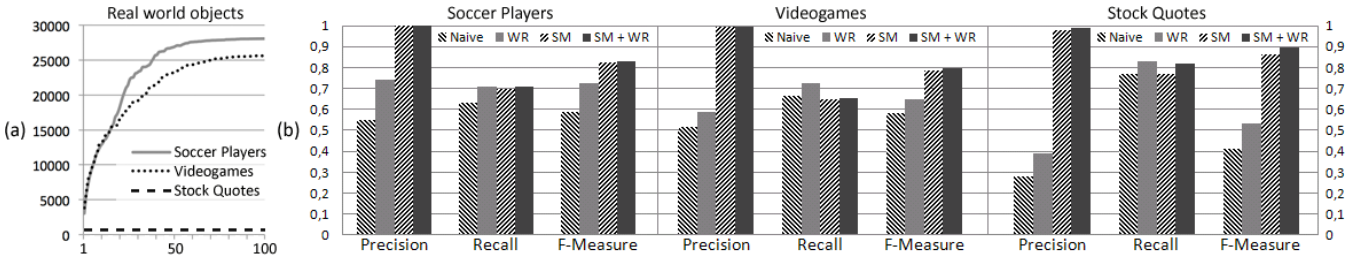

Figure 5: (a) Growing of the number of real-world objects over the number of sources. (b) Precision, Recall, and F-measure of the mappings of four different executions: naive matching, naive matching with wrapper refinement (WR), SplitAndMerge (SM), SplitAndMerge with wrapper refinement (SM+WR).

hidden relation. The mappings and the extraction rules produced by our system allow us to build an integrated view of its whole extension by accumulating information from many sources, thus obtaining more information than actually exposed by each participating source. To give an example, consider the objects and the 8 attributes reported for the Soccer and the Finance domains in Table 2: the hidden relations for these entities contain at most 224k values (28k objects × 8 attributes) and 4.6k (576×8) values, respectively. In the finance domain, a single source covers on average about 3.2k values (70% of the total), while the integrated view over the 100 sources reaches 4.1k values (90% of the total). More interestingly, in the soccer domain, a single source covers on average only 1.4k values (1% of the total), while the integrated view reaches 134k values (71%). As for the same object and attribute different values are provided by distinct sources, conflicting values can arise. To overcome this issue recent works, such as [11], represent suitable solutions.

5. RELATED WORK

Information Extraction DIPRE [5] represents a pioneering technique to extract relations from the Web. Starting from a bunch of seed pairs (e.g., author-book), it collects 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 paper motivated several web information extraction projects to develop effective techniques for the extraction of facts (binary predicates, e.g., born-in⟨scientist, city⟩) from large corpora of web pages (e.g. [2]). Web information extraction techniques mainly infer lexico-syntactic patterns from textual descriptions, but, as discussed in [6], they cannot take advantage of the structures available on the Web, as they do not elaborate data that are embedded in HTML templates.

Data Integration The bootstrap of data integration architecture for structured data on the Web is the subject of [12]. However, the proposed integration techniques are based on the availability of attribute labels; on the contrary, our approach aims at integrating unlabeled data from web sites and automatically infers the labels whenever they are encoded in the HTML templates. The exploitation of structured web data is the primary goal of WebTables [8], which concentrates on data published in HTML tables. Compared to information extraction approaches, WebTables extracts relations with involved relational schemas but it does not address the issue of integrating the extracted data. Cafarella *et al.* have described a system to populate a probabilistic database with data extracted from the Web [6]. However, the data are retrieved by TextRunner [2], an information extraction system that is not targeted to data rich web pages as ours. Octopus [7] and Cimple [13] support users in the creation of data sets from web data by means of a set of operators to perform search, extraction, data cleaning and integration. Although such systems have a more general application scope than ours, they involve users in the process, while our approach is completely automatic.

Wrapper Inference RoadRunner [10] and ExAlg [1] propose automatic inference techniques to create a wrapper from a collection of pages generated by the same template. An approach related to ours is developed in the TurboWrapper project [9], which introduces a composite architecture including several wrapper inference systems (e.g., [1, 10]). By means of an integration step of their output it aims at improving the results of the single participating systems taken separately. However, compared to our solution, it makes stronger domain dependent assumptions for the integration step since it does not consider the redundancy of information that occurs at the instance level.

6. REFERENCES

[1] A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *SIGMOD*, 2003.

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

[3] A. Bilke and F. Naumann. Schema matching using duplicates. In *ICDE*, 2005.

[4] L. Blanco, V. Crescenzi, P. Merialdo, and P. Papotti. Supporting the automatic construction of entity aware search engines. In *WIDM*, 2008.

[5] S. Brin. Extracting patterns and relations from the World Wide Web. In *WebDB*, 1998.

[6] M. J. Cafarella, O. Etzioni, D. Suciu. Structured queries over web text. *IEEE Data Eng. Bull.*, 29(4):45–51, 2006.

[7] M. J. Cafarella, A. Y. Halevy, N. Khoussainova. Data integration for the relational web. *PVLDB*, 2(1), 2009.

[8] M. J. Cafarella, A. Y. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. Webtables: exploring the power of tables on the web. *PVLDB*, 1(1):538–549, 2008.

[9] S.-L. Chuang, K. C.-C. Chang, C. X. Zhai. Context aware wrapping: Synchronized data extraction. In *VLDB*, 2007.

[10] V. Crescenzi, G. Mecca, and P. Merialdo. RoadRunner: Towards automatic data extraction from large Web sites. In *VLDB*, 2001.

[11] X. L. Dong, L. Berti-Equille, and D. Srivastava. Integrating conflicting data: The role of source dependence. *PVLDB*, 2(1):550–561, 2009.

[12] A. D. Sarma, X. Dong, and A. Y. Halevy. Bootstrapping pay-as-you-go data integration systems. In *SIGMOD*, 2008.

[13] W. Shen, P. DeRose, R. McCann, A. Doan, and R. Ramakrishnan. Toward best-effort information extraction. In *SIGMOD*, 2008.