Intelligent Self-Repairable Web Wrappers

Emilio Ferrara[1] and Robert Baumgartner[2]

[1] Dept. of Mathematics, University of Messina, V. Stagno d'Alcontres 31, 98166 Italy
emilio.ferrara@unime.it
[2] Lixto Software GmbH, Favoritenstrasse 16/DG, 1040 Vienna, Austria
robert.baumgartner@lixto.com

Abstract. The amount of information available on the Web grows at an incredible high rate. Systems and procedures devised to extract these data from Web sources already exist, and different approaches and techniques have been investigated during the last years. On the one hand, reliable solutions should provide robust algorithms of Web data mining which could automatically face possible malfunctioning or failures. On the other, in literature there is a lack of solutions about the maintenance of these systems. Procedures that extract Web data may be strictly interconnected with the structure of the data source itself; thus, malfunctioning or acquisition of corrupted data could be caused, for example, by structural modifications of data sources brought by their owners. Nowadays, verification of data integrity and maintenance are mostly manually managed, in order to ensure that these systems work correctly and reliably. In this paper we propose a novel approach to create procedures able to extract data from Web sources – the so called *Web wrappers* – which can face possible malfunctioning caused by modifications of the structure of the data source, and can automatically repair themselves.

Keywords: Web data extraction, wrappers, automatic adaptation

1 Introduction

The actual panorama of distribution of information through the Web depicts a clear situation: there is an incredible amount of data delivered under the form of Web data sources and a correspondingly need of capability of mining these information in a reliable and efficient way. Mining information from Web sources is a task which obviously can be useful in several different area of the knowledge. Moreover, this topic interests both the academia and the enterprises. For example, consider the following scenarios: i) a research group which needs to acquire a dataset of information delivered through online services, say for example an online database publishing, day by day, information about the mapping of some genes; ii) a company, for which it is essential for marketing and product placement to monitor the trends of pricing of services offered by its competitors, provided through the Web. Both the two actors need to extract, possibly, a huge amount of data during an extend period of time (e.g., months), at regular intervals (say, each day). One important aspect in both the cases is the reliability and

the quality of data extracted. It is utterly important that acquired information is correct, because the research group can not accept corrupted data and the comparison with competitors will fail in case of bad product data.

These two examples highlight common requirements in the panorama of Web data mining, and depict different related problems. Although in literature some techniques to design systems for the extraction of data from Web sources have been presented, there is a lack of work in the area of their maintenance. An ample number of questions and problems related to the possibility of automatize the process of maintenance are still uncovered. This work tries to focus on some aspects related to the maintenance of these systems. We first introduce the theoretical background required to create intelligent procedures of Web data extraction. Then, we explain how to face malfunctioning likely to happen during the extraction process, for example caused by modifications in the structure of the data source. The second point in particular is the main focus of this work. Let us contextualize this problem: essentially there exist two different approaches to extract information from Web sources. The first one relies on machine learning platforms [5]; a system analyzes, possibly, huge amount of positive and negative examples during a training period, and, then, it infers some set of rules that makes it able to perform its tasks in the same domain or Web site. Different approaches rely on logic-based algorithms which analyze the structure of the data source and induct some procedures to extract required information exploiting structural characteristics of the Web source to identify and find required data. The second approach utilizes the knowledge a human can bring in about a particular site or domain. The wrapper is generated in a way that the human creates the rules and navigation paths together with the system in a supervised and interactive way. Still, the system can assist the wrapper designer and offer possibilities that make the wrapper execution as robust as possible, even in case of structural changes. From now, in this work we assume that the platform we are going to describe and improve adopts the latter philosophy.

Organization of the paper We describe related work in Section 2. In Section 3, the algorithmic background is introduced, describing an efficient tree matching technique. Section 4 covers the design of robust and adaptable procedures of Web data extraction, the so called *intelligent self-repairable Web wrappers*. Then, in Section 5 we describe the adaptation process during wrapper execution. We explain how these procedures can automatically, in an autonomous way, face malfunctioning, trying to adapt themselves to modifications which possibly caused problems. A prototype has been implemented on top of a state-of-the-art extraction platform, the Lixto Visual Developer. Performance of this system are shown in Section 6, by means of precision and recall scores. Section 7 concludes summarizing our main achievements and depicting some future work.

2 Background and Related Work

We split related literature in three main topics: i) Web data extraction systems; ii) Maintenance and related problems; iii) tree matching algorithms.

Web data extraction systems The work related to systems of Web information extraction is manifold but well depicted by several surveys. Laender et al. [13] provided the first rigorous taxonomical classification of Web data extraction systems. Kushmerick [11] classified several finite-state approaches to generate wrappers, such as the wrapper induction, natural language processing approaches and hidden Markov models. Sarawagi [17] provided the most comprehensive survey on the information extraction panorama. This work covers different existing techniques explaining several approaches. In the last years, first Baumgartner et al. [1] and later Ferrara et al. [8] provided two different surveys on the discipline of Web data extraction. The first is mainly addressed to practitioners, the latter focuses on application fields of this discipline.

Maintenance and related problems Although some interesting work, we can identify a general lack of solutions provided in the area of the Web wrapper maintenance. Kushmerick [12, 10] for first introduced the concept of wrapper maintenance as the process of verifying the correct functioning of the data extraction procedures and manually, automatically or in a semi-automatic way, intervene in case of malfunctioning. Lerman and Minton [14], instead, faced both the problems of verifying the correctness of data extracted by a wrapper and eventually try to repair it. Their approach is a mix of machine learning techniques. Another approach based on machine learning has been provided by Chidlovskii [4]; he described a system which can automatically classify Web pages in order to extract information from those pages which can be handled adopting both conventional extraction rules and ensemble methods of machine learning, such as the content features analysis. Meng et al. [15] developed the SG-WRAM (Schema-Guided WRApper Maintenance) slightly modifying the perspective of Web wrappers generation, observing that changes in Web pages, even substantial, always preserve syntactic features (i.e., syntactic characteristics of data items like data patterns, string lengths, etc.), hyperlinks and annotations (e.g., descriptive information representing the semantic meaning of a piece of information in its context). Finally, another heuristic approach has been presented by Raposo et al. [16]; they adopted a collected sample of positive labeled examples during the normal execution of the wrappers, to be exploited in case of malfunctioning, in order to re-induct the broken wrapper ensuring a good accuracy of the process.

Tree Matching In general, the process of comparing the structure of two trees is a well-known classic problem. The possibility of transforming a tree into another one, through a sequence of (possibly different) operations, is another well-known algorithmic challenge, namely the *tree editing* problem. The minimum number of elementary transformations, such as adding/removing nodes, relabeling nodes or moving nodes, represents the *distance* between two trees. This value can be

used to represent the measure of dissimilarity between two trees. The tree edit distance problem is a well-known NP-hard problem [3]. Several approximate solutions have been advanced during the years; the most appropriate algorithm to face the problem of matching up similar trees, has been suggested by Selkow [18]. This technique relies on the concept of finding isomorphic elements present in both the two compared trees, implementing a light-weight recursive top-down resolution during which the algorithm evaluates the position of nodes to measure the degree of isomorphism between them, analyzing and comparing their sub-trees. Different versions of this algorithm exist; each of them presents some optimizations. Ferrara and Baumgartner [6, 7] so as Yang [19] adopt *weights*, obtaining a variant of this algorithm with the capability of discovering clusters of similar sub-trees. An interesting evaluation of the simple tree matching and its weighted version, presented by Kim et al. [9], has been performed exploiting these two algorithms for extracting information from HTML Web pages. These optimized algorithms underly the design of our self-repairable Web wrappers.

3 The Tree Matching Algorithm

This work relies on some assumptions: i) Web pages are represented by using DOM trees, as the HTML standard suggests [3]; ii) it is possible to identify elements within a DOM tree by using the XPath language [4]; iii) the logics of XPath underly the functioning of Web wrappers (this is further explained in following sections and in [1, 2]). Given these milestones, the main idea of our approach is to compare two trees, one representing the original Web page and another representing the page after that some modifications occurred. This is practical in order to automatize the adaptive process of automatic repairing of our wrappers. To do so, we utilize a variant of the seminal Simple Tree Matching (STM) [18], optimized by Ferrara and Baumgartner [6, 7]. Let *d(n)* be the degree of a node n (i.e., the number of first-level children); let $T(i)$ be the i-*th* sub-tree of the tree rooted at node T; let $t(n)$ be the number of total siblings of a node n including itself. The *Weighted Tree Matching* here described (see Algorithm 1) optimizes the simple tree matching, for our specific domain.

4 Web Wrappers

In supervised and interactive wrapper generation, the application designer is in charge of deciding how to characterize Web objects that are used for traversing the Web and for extracting information. It is one of the most important aspects of a wrapper to be resilient against changes (both changes over time and variations of similarly structured pages), and parts of the robustness of a data extractor depend on how the application designer configures it. However, it is crucial that the wrapper generation system assists the wrapper designer and suggests how

[3] http://www.w3.org/TR/DOM-Level-2-HTML/html.html
[4] http://www.w3.org/TR/xpath/

Algorithm 1 WeightedTreeMatching($T^{'}$, $T^{''}$)

```
1: if T' has the same label of T'' then
2:    m ← d(T')
3:    n ← d(T'')
4:    for i = 0 to m do
5:       M[i][0] ← 0;
6:    for j = 0 to n do
7:       M[0][j] ← 0;
8:    for all i such that 1 ≤ i ≤ m do
9:       for all j such that 1 ≤ j ≤ n do
10:         M[i][j] ← Max(M[i][j − 1], M[i − 1][j], M[i − 1][j − 1] + W[i][j]) where
            W[i][j] = WeightedTreeMatching(T'(i − 1), T''(j − 1))
11:   if m > 0 AND n > 0 then
12:      return M[m][n] * 1 / Max(t(T'), t(T''))
13:   else
14:      return M[m][n] + 1 / Max(t(T'), t(T''))
15: else
16:   return 0
```

to make the identification of Web objects and trails through Web sites as stable as possible.

4.1 Robust XPath Generation and Fall-back Strategies

In Lixto Visual Developer (VD) [2], a number of mechanisms are offered to create a resilient wrapper. During recording, one task is to generate a robust XPath or regular expression, interactively and supported by the system. During wrapper generation, in many cases only one labeled example object is available, especially in automatically recorded deep Web navigation sequences. In such cases, efficient heuristics in XPath generation and fallback strategies during replay, are required. Typical heuristics during recording for reliably identifying such single Web objects include:

- Generalization of a chosen XPath by using form properties, element properties, textual properties and formatting properties. During replay, these ingredients are used as input for an algorithm that checks in which constellation to best apply this property information to satisfy the integrity constraints imposed on a rule (e.g. as result a single instance is required).
- DOM Structural Generalization – starting from the full path, several generalized paths are created, using only characteristic elements and characteristic element sequences. A number of stable anchor points are identified and stored, from which relative paths to this object are created. Typical stable anchor points are identified automatically and include, e.g., the outermost table structure and the main content area (being chosen upon factors such as the longest content).

– Positional information is considered if the structurally generalized paths identify more than one element. In this case, during execution, variations of the XPath generated with this "index heuristics" are applied on the active Web page, removing indexes until the integrity constraints of the current rule are satisfied.
– Attributes and properties of elements are taken into account, in particular of the element of choice, but we also consider ancestor attributes if the element attributes are not sufficient.
– Attributes that make an element unique are preferred, i.e. similar elements are checked for distinguishing criteria.
– Attribute Values are considered, if attribute names are not sufficient. Attribute Value Fragments are considered, if attribute values are not sufficient (using regular expressions).
– The ID attributes are used as far as possible. If an ID is unique and meaningful for characterizing an element it is considered in the fallback strategies with a high weight.
– Textual information and label information is used, only if explicitly turned on (since this might fail in case of a language switch).

The output of the heuristic step is a "best XPath" shown to the wrapper designer, and a set of XPath expressions and priorities regarding when to use which fallback strategy, stored in the configuration. Figure 1 illustrates which information is stored by the system during recording. In this case, a drop down was selected by the application designer, and the system decided that the "id" attribute is the most reliable one and is chosen as best XPath. If this evaluation fails, the system will apply heuristics based on the (in this example three) stored fallback XPaths, which mainly exploit form and index properties. In case one of the heuristics generates results that do not invalidate the defined integrity constraints, these Web objects are considered as result.

During generation of rules (e.g. "extract") and actions (e.g. "click"), the wrapper designer imposes constraints on the results to be obtained, such as:

– Cardinality Constraints: restrictions on the number of results, e.g. exactly one element or at least one element must be matched.
– Data Type Constraints: restrictions on the data type of a result, e.g. a result must be of type integer or match a particular regular expression.

Constraints can be defined individually per rule and action, or defined globally by using a schema on the output data model.

4.2 Configuring Adaptable Wrappers

The procedures described in the previous section do not adapt the wrapper, but address situations in which the initially chosen XPath does no longer match and simply try different ones based on this one. In the configuration of wrapper adaptation, we go one step beyond: on the one hand we exploit tree and string

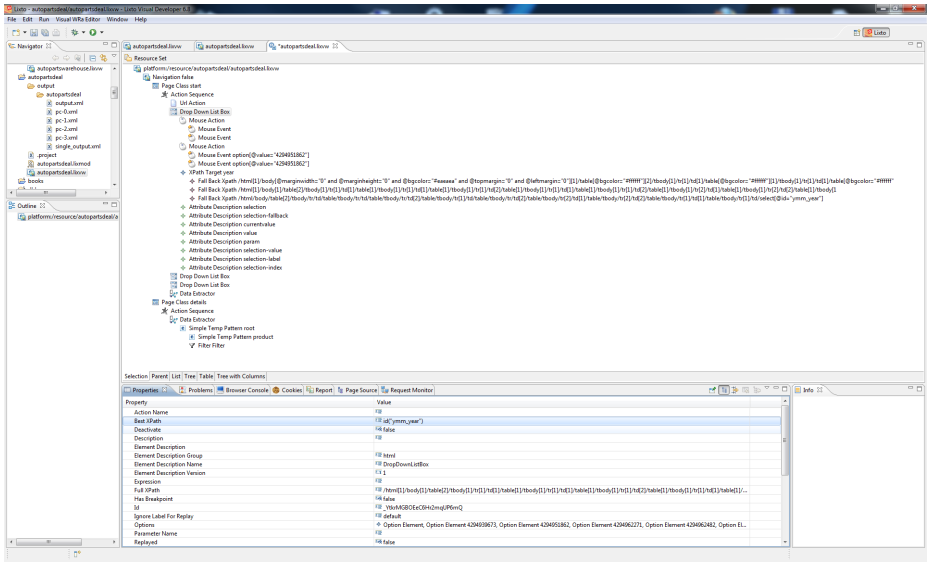

Fig. 1. Robust Web Object Detection in Visual Developer.

similarity techniques to find the most similar Web object(s) on the new page, and on the other hand, in case the adaptation is triggered, the wrapper is changed on the fly using the new configuration created by the adaptation algorithms.

As before, integrity constraints can be imposed on extraction and navigation rules. Moreover, the application designer can choose whether to use wrapper adaptation on a particular rule in case the constraints are violated during runtime. When adaptation is chosen, alternatively to using XPath-based means to identify Web objects we store the actual result subtree. In case of HTML leaf elements, which are usually the elements under consideration for navigation actions, we instead store the tree rooted at the *n-th* ancestor of the element, and the additional fact where the result element is located within this tree. In this way, tree matching can also be exploited for HTML leaf elements.

Wrapper designers can choose between various similarity measures: this includes in particular the Simple Tree Matching algorithm [18] and the Weighted Tree Matching algorithm described in Section 3. In future, further algorithms will extend the capabilities of the tool, e.g. a bigram-based tree matching that is capable to deal with node permutations in a more favorable fashion. In addition to the similarity function, one can choose certain parameters, e.g. whether to use the HTML element name as node label or instead to use spelling attributes such as *class* and *id* attributes. Figure 2 illustrates the configuration of wrapper adaptation in Visual Developer.

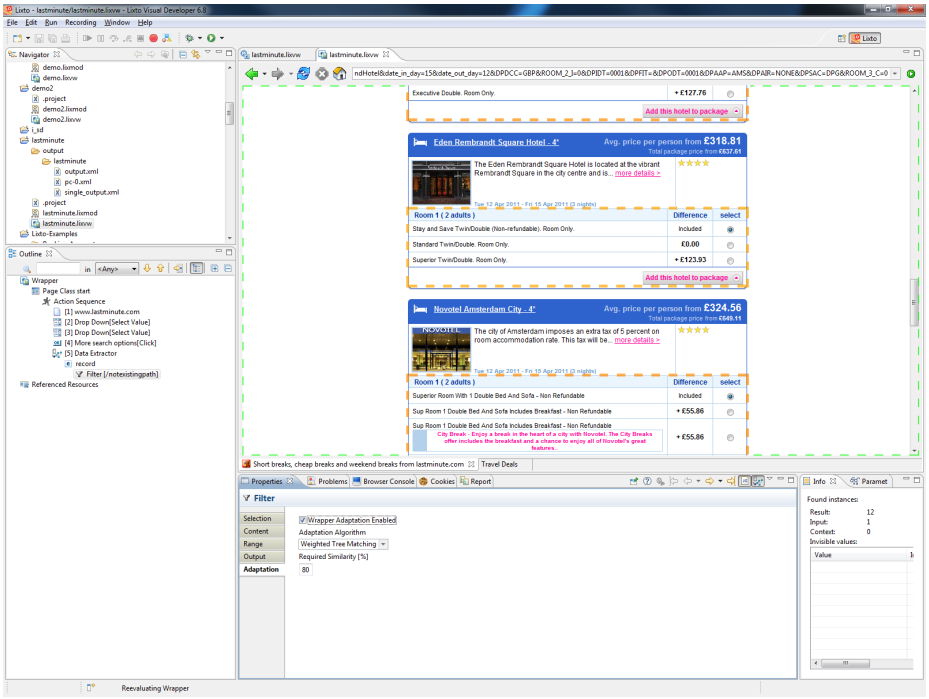

Fig. 2. Configuration of Wrapper Adaptation in Lixto VD.

5 Automatic Wrapper Adaptation

5.1 Self-repairing rules

Figure 3 describes the adaptation process. The wrapper adaptation process is triggered upon violation of defined constraints. In case in the initial wrapper an element is detected with an XPath, the adaptation procedure substitutes this by storing the subtree of a matched element. In case the wrapper definition already stores the example tree, and the similarity computation returns results that violate the defined constraints, the threshold is lowered or raised until a perfect match is generated.

During runtime, the stored tree is compared to the elements on the new page, and the best fitting element(s) are considered as extraction results. During configuration, wrapper designers can choose an algorithm (such as the Weighted Tree Matching), and a similarity threshold. The similarity threshold can be constant, or defined to be within an interval of acceptable thresholds. During execution, various thresholds within the allowed range are considered, and the one generating the best fit with respect to the defined constraints is chosen.

As a next step, the stored tree is refined and generalized that it maximizes the matching value for both the original subtree and the new trees, reflecting the changes of a Web page over time. This generalization process generates a simple

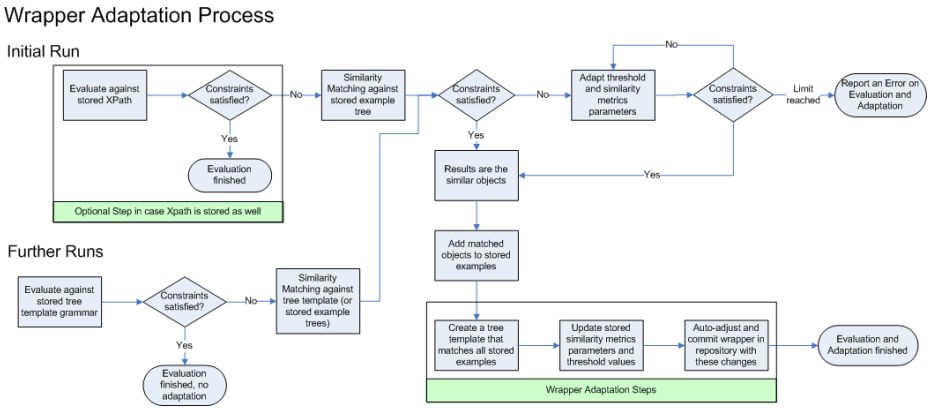

Fig. 3. Wrapper Adaptation Process.

tree grammar, a "tree template" that is allowed to use occurrence indicators (one or more element, at least one element, etc.) and optional depth levels. In further runs, the tree template is compared against the sub trees of an active Web page during execution. First, the algorithm checks which trees on the new page satisfy the tree template. In case the results are within the defined integrity constraints, no further action is taken. In case the results are not satisfying, the system searches for most similar trees based on the defined distance metrics; in this case, the wrapper is auto-adapted, the tree template is further refined and the threshold or threshold interval is automatically re-adjusted. At the very end of the process, the corrected wrapper is stored in the wrapper repository and committed to a versioning system to keep track of all changes.

5.2 Wrapper Re-Induction

In practice, single adaptation steps of rules and actions are embedded into the whole execution process of a wrapper and the adapted wrapper is stored in the repository after all adaptation steps have been concluded. The need for adapting a particular rule influences the further execution steps.

Usually, wrapper generation in VD is a hierarchical top-down process – e.g. first, a "hotel record" is characterized, and inside the hotel record, entities such as "rating" and "room types". To define a rule to match such entities, the wrapper designer visually selects an example and together with system suggestions generalizes the rule configuration until the desired instances are matched. To support the automatic adaptation process during runtime, as described above, the wrapper designer further specifies what it means that extraction failed. In general, this means wrong or missing data, and with integrity constraints one can give indications how correct results look like. The upper half of Figure 4 summarizes the wrapper generation.

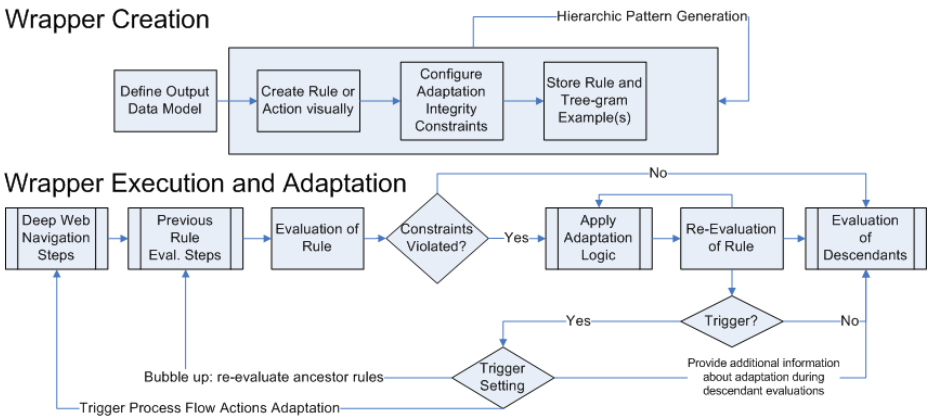

Fig. 4. Diagram of the Web wrapper creation, execution and maintenance flow.

During wrapper creation, the application designer provides a number of configuration settings to this process. This includes:

- Threshold Values.
- Priorities/Order of Adaptation Algorithms used.
- Flags of the chosen algorithm (e.g. using HTML element name as node label, using id/class attributes as node labels, etc.).
- Triggers for bottom-up, top-down and process flow adaptation bubbling.
- Whether stored tree-grams and XPath statements are updated based on adaptation results to be additionally used as inputs in future adaptation procedures (reflecting and addressing regular slight changes of a Web page over time).

Triggers in Adaptation Settings can be used to force adaptation of further fragments of the wrapper as depicted in the lower half of Figure 4.

- Top-down: forcing adaptation of all/some descendant rules (e.g. adapt the "price" rule as well to identify prices within a record if the "record" rule was adapted).
- Bottom-up: forcing adaptation of a parent rule in case adaptation of a particular rule was not successful. Experimental evaluation pointed out that in such cases it is often the problem that the parent rule already provides wrong or missing results (even if matched by the integrity constraints) and has to be adapted first.
- Process flow: it might happen that particular rule matches can no longer detected because the wrapper evaluates on the wrong page. Hence, there is the need to use variations in the deep web navigation actions. In particular, a simple approach explored at this time is to use a switch window or back step action to check if the previous window or another tab/popup provides the required information.

		Simple Tree Matching			Weighted Tree Matching		
		Precision/Recall			Precision/Recall		
Scenario	thresh.	tp	fp	fn	tp	fp	fn
Delicious	40%	100	4	-	100	-	-
Ebay	85%	200	12	-	196	-	4
Facebook	65%	240	72	-	240	12	-
Google news	90%	604	-	52	644	-	12
Google.com	80%	100	-	60	136	-	24
Kelkoo	40%	60	4	-	58	-	2
Techcrunch	85%	52	-	28	80	-	-
Total	-	1356	92	140	1454	12	42
Recall	-	90.64%			97.19%		
Precision	-	93.65%			99.18%		
F-Measure	-	92.13%			98.18%		

Table 1. Experimental performance evaluation in real world scenarios.

6 Performances Measurement

For our initial performance evaluation we tested the robustness of our Wrappers against real world use-cases. Actual areas of interest for Web data extraction problems include social networks, retail market and Web communities. We defined a total of 7 scenarios and designed 10 adaptive wrappers each. Results, by means of precision, recall and F1-score, are as shown in Table 1. Column *thresh.* represents the fixed threshold value; *tp*, *fp* and *fn* summarize true and false positive, and false negative, respectively. Performance obtained by using *simple* and *weighted* tree matching are good; these algorithms are definitely viable solutions to our initial purpose and provide high degree of reliability (F-Measure $> 90\%$).

7 Conclusions and Future Work

In literature, several implementations of systems to extract data from Web sources have been presented, but there is a lack of solutions about their maintenance. This paper tries to address this problem, describing adaptive techniques to make Web data extraction systems, based on wrappers, self-maintainable, adopting algorithms optimized to this purpose. So, enhanced Web wrappers become able to recognize structural modifications of Web sources and to adapt their functioning accordingly. Characteristics of our self-repairable solution are discussed in details, providing first experimental results to evaluate its robustness. More experimentation has to come in the next future.

Moreover, as for future work, additional algorithms would be included in order to improve the capabilities of the adaptation feature; in particular, a viable idea could be to generalize a bigram-based tree matching algorithm capable of dealing with node permutations in a more efficient way with respect to Simple Tree Matching based algorithms adopted as to date. Similarly, the Jaro-Winkler

distance could be adapted to our tree matching problem in order to better reflect missing or added node levels, so as improving performance of our adaptation process. Finally, the tree-grammar could be extended to classify different topologies of templates (those frequently adopted by Web pages), in order to define several standard protocols of automatic adaptation, to be adopted in specific contexts.

References

1. Baumgartner, R., Gatterbauer, W., Gottlob, G.: Web data extraction system, pp. 3465–3471. Springer-Verlag New York, Inc (2009)
2. Baumgartner, R., Gottlob, G., Herzog, M.: Scalable web data extraction for online market intelligence. Proceedings of the VLDB Endowment 2(2), 1512–1523 (2009)
3. Bille, P.: A survey on tree edit distance and related problems. Theoretical computer science 337(1-3), 217–239 (2005)
4. Chidlovskii, B.: Automatic repairing of web wrappers by combining redundant views. In: Proceedings of the 14th International Conference on Tools with Artificial Intelligence. pp. 399–406. IEEE (2003)
5. Esposito, F., Malerba, D., Di Pace, L., Leo, P.: A machine learning approach to web mining. AI* IA 99: Advances in Artificial Intelligence pp. 190–201 (2000)
6. Ferrara, E., Baumgartner, R.: Automatic wrapper adaptation by tree edit distance matching. Combinations of Intelligent Methods and Applications pp. 41–54 (2011)
7. Ferrara, E., Baumgartner, R.: Design of automatically adaptable web wrappers. In: ICAART '11: Proceedings of the 3rd International Conference on Agents and Artificial Intelligence. pp. 211–217 (2011)
8. Ferrara, E., Fiumara, G., Baumgartner, R.: Web data extraction, application and techniques: A survey. Technical Report (2011)
9. Kim, Y., Park, J., Kim, T., Choi, J.: Web information extraction by HTML tree edit distance matching. In: Proceedings of the International Conference on Convergence Information Technology. pp. 2455–2460. IEEE (2008)
10. Kushmerick, N.: Wrapper verification. World Wide Web 3(2), 79–94 (2000)
11. Kushmerick, N.: Finite-state approaches to Web information extraction. Extraction in the Web Era pp. 77–91 (2003)
12. Kushmerick, N., et al.: Regression testing for wrapper maintenance. In: Proceedings of the National Conference on Artificial Intellingece. pp. 74–284 (1999)
13. Laender, A., Ribeiro-Neto, B., da Silva, A., Teixeira, J.: A brief survey of web data extraction tools. ACM Sigmod Record 31(2), 84–93 (2002)
14. Lerman, K., Minton, S., Knoblock, C.: Wrapper maintenance: A machine learning approach. Journal of Artificial Intelligence Research 18(1), 149–181 (2003)
15. Meng, X., Hu, D., Li, C.: Schema-guided wrapper maintenance for web-data extraction. In: Proceedings of the 5th ACM international workshop on Web information and data management. pp. 1–8. ACM (2003)
16. Raposo, J., Pan, A., Alvarez, M., Hidalgo, J.: Automatically generating labeled examples for web wrapper maintenance. In: Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence. pp. 250–256 (2005)
17. Sarawagi, S.: Information extraction. Foundations and Trends in Databases 1(3), 261–377 (2008)
18. Selkow, S.: The tree-to-tree editing problem. Information processing letters 6(6), 184–186 (1977)
19. Yang, W.: Identifying syntactic differences between two programs. Software: Practice and Experience 21(7), 739–755 (1991)