WEWRA: An algorithm for Wrapper Verification

Charalampos E. Tsourakakis Georgios Paliouras†

March 2009
CMU-ML-09-100

VEWRA: An algorithm for Wrapper Verification

Charalampos E. Tsourakakis[*] **Georgios Paliouras**[†]

March 2009
CMU-ML-09-100

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[*]Machine Learning Department, Carnegie Mellon University, Pittsburgh, PA, USA
[†] Institute of Informatics & Telecommunications, NCSR "Demokritos", 15310, Ag. Paraksevi, Attiki, Greece, paliourg@iit.demokritos.

Abstract

Web wrappers play an important role in extracting information from distributed web sources and subsequently in the integration of heterogeneous data. Changes in the layout of web sources typically break the wrapper, leading to erroneous extraction of infomation. Monitoring and repairing broken wrappers is an important hurdle for data integration, since it is an expensive and painful procedure. In this paper we present VEWRA, a new approach to wrapper verification, which improves the successful family of trainable content - based methods. Compared to its predecessors, the new method aims to capture not only the syntactic patterns but the correlations that exist among them due to the underlying semantics of the extracted information. Experiments show that our method achieves excellent performance, being always better or equal than DATAPROG, the state-of-art related work.

Keywords: wrapper verification, wrapper maintenance, web wrappers

1 Introduction

The Web is an enormous source of information and increasing at a fast pace. However, the information that is made available in Web pages is not always easily accessible, particularly for machines. The extraction and integration of information from distributed web sources found on the web has attracted significant research interest over the last years (e.g [2], [3], [5], [6], [8], [13]). This observation has also led to the very active initiative of the Semantic Web ([1]), which aims to make the Web more machine-friendly.

The scarcity of machine-readable information on the Web is becoming paradoxical when considering the proportion of dynamically - produced content. In [19] it was estimated that 80% of the Web's content was generated from databases. On the other hand, the automated generation of Web pages leads to the semi-structured nature of Web content, which facilitates its fairly reliable parsing, based on the HTML formatting tags. The parsing programs are usually called wrappers and are based on simple regular patterns, which are specific to each particular site that the machine is expected to access. Due to the limited scope of these extraction patterns, the manual construction and maintenance of Web wrappers is not scalable ([14]). For this reason, the study of trainable and adaptive wrappers, with the use of machine learning methods, has attracted significant interest.

In this context, we focus here on the task of automatic maintenance of Web wrappers and in particular on the identification of potential changes to the layout of Web pages which lead to erroneous extraction of information. This task is known as *wrapper verification* and is central to the automation of wrapper maintenance, as it can trigger the process of wrapper updating, also known as *wrapper re-induction*. Since integrating information from different web sources relies heavily on web wrappers[1], *wrapper verification* is an important practical problem. In this work, we present VEWRA, a novel wrapper verification system. Figure 1 shows the performance of our algorithm versus DATAPROG [2], the state-of-art content based algorithm (to the best of our knowledge) proposed in [12] on the same dataset. Evaluating both algorithms using the widely used measures in information retrieval, we see that VEWRA outperforms DATAPROG, by having always better or equal performance.

The main contributions of this work are the following:

- **VEWRA**: A new wrapper verification algorithm, which aims to capture successfully not only the syntactic regularities of the extracted data but also the correlations among them. The key assumption of VEWRA is that given the semantics of the extracted data there is a non-uniform probability distribution among the possible patterns. This is achieved through several novel ideas, such as groups of associated syntactic patterns and an intuitive penalty system. Furthermore, on the contrary to prior work which relies heavily on HTML token density as an indicator of layout change, we remove this reliance since it can be a potential source of false alarms as justified in section 2.

- **Extensive Experimentation**: We used the datasets of the previous work ([9], [12]) to validate our method and obtain a clear picture of the performance of VEWRA compared to the state-of-art content-based methods.

The rest of this paper is structured as follows. In section 2 we illustrate the concepts related to wrapper maintenance and we briefly present previous work on the wrapper verification problem, emphasizing on content-based methods. In section 3 we describe our new method, VEWRA. In section 4 we present initial

[1]Although XML is supposed to alleviate data integration problems, relatively few web sources for the time being support this data format. Therefore, HTML web sources are still dominating.

[2]For convenience, we will call this wrapper verification system DATAPROG, even if DATAPROG is the algorithm that learns syntactic patterns and not the whole verification system.

experimental evidence of the good performance of our method and finally in section 5 we conclude and present some perspectives for future work.

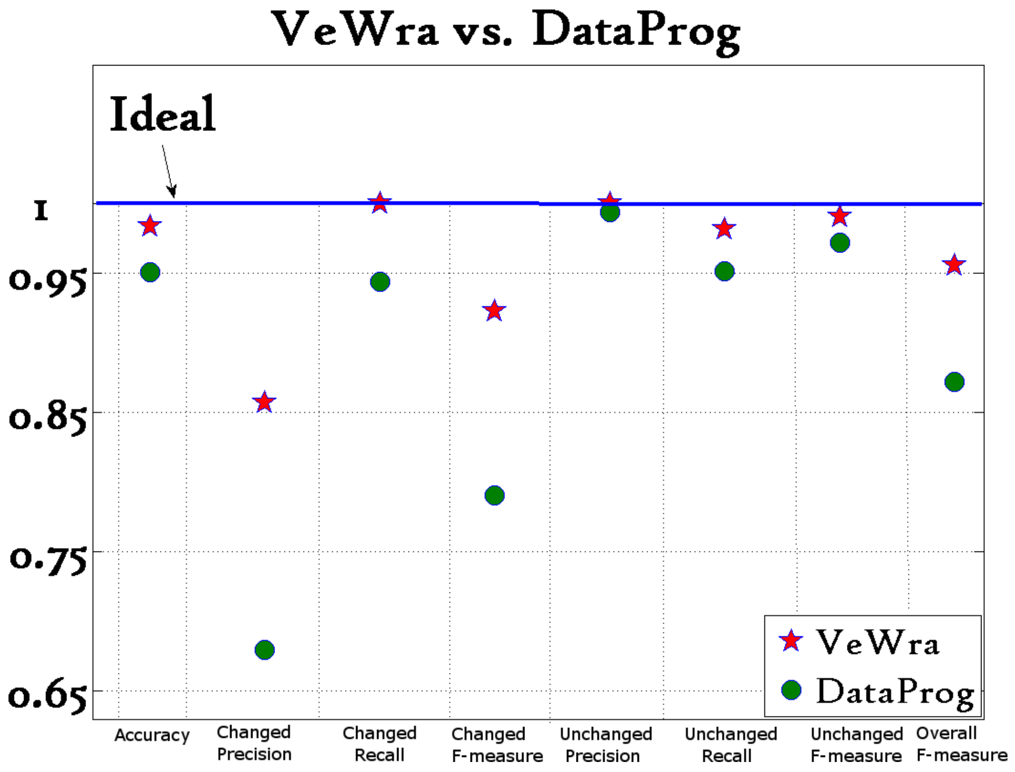

Figure 1: Performance of VEWRA versus DATAPROG, the state-of-art wrapper verification algorithm. Our method achieves equal or better performance for all widely used measures.

2 Background

2.1 Main Concepts

The envisioned automated wrapper construction and verification process, that is based on learning from examples is illustrated in figure 2. According to this process, the training examples are initially provided manually to a wrapper induction system, which produces the first version of the wrapper. Following that point, wrapper verification and wrapper re-induction become responsible for generating new training examples, resulting in improved versions of the wrapper. Wrapper verification is a very important subproblem of data integration from web sources: one can easily imagine extracting data from hundreds of different web sources, using one wrapper per each. The ability to identify when a wrapper is broken leads to very significant savings of time and makes wrapper-based data integration systems viable.

In order to illustrate the aforementioned concepts we present a simple example of what a wrapper is and why maintenance is needed. Table 1 presents the source of a Web page (top), the rendered content (top center), the wrapper that extracts the `movie title` and `director` attributes respectively (bottom center), and the extracted information (bottom). If the layout of the page changes, as shown in table 2,

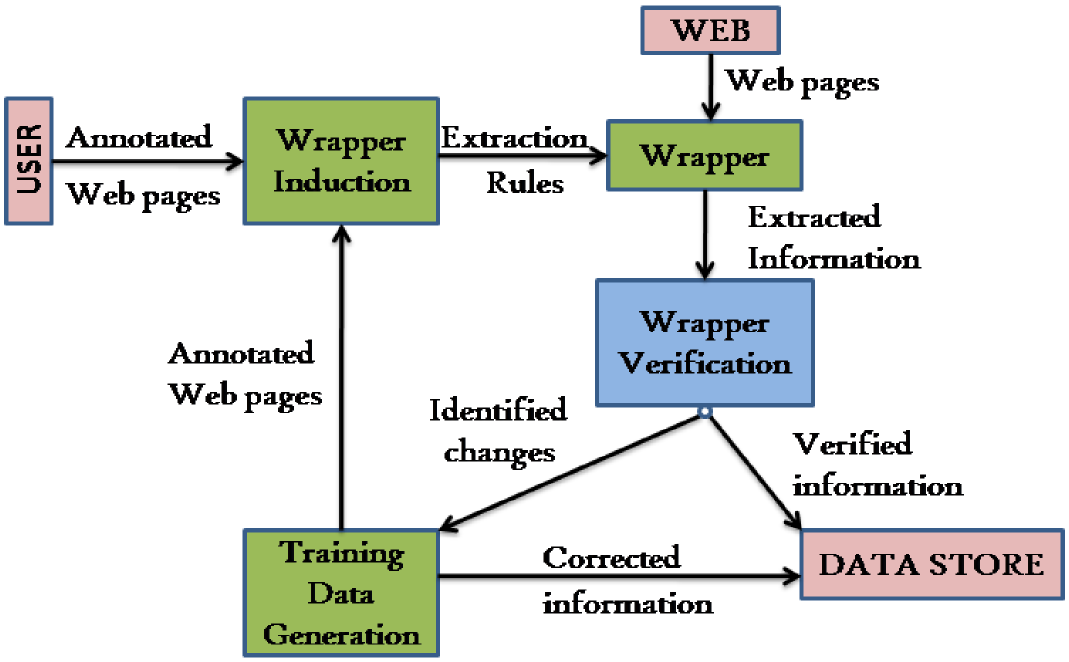

Figure 2: Automated wrapper maintenance.

the wrapper will cease to extract correct information from the Web site. In particular, information for the attribute `movie title` would be incorrectly extracted, while for the attribute `director` no information at all would be extracted.

In such a situation, a wrapper verification system should recognize that wrapper *ccwrap1* is broken and the wrapper reinduction system should produce examples and train a new wrapper. Such a wrapper could be *ccwrap2*, shown in table 2.

Effectively, what has changed between *ccwrap1* and *ccwrap2* is the set of begin and end *landmark* patterns that delimit the extracted fields. Thus, the goal of wrapper induction is to learn from appropriate examples these landmarks, which in more realistic cases could take the form of regular expressions (e.g [11], [17], [4]).

The main goal of our work was to develop a wrapper verification method that will identify reliably and accurately when a wrapper is broken. More specifically, we present here a content-based verification method, i.e., a method that is based on the extracted information to test whether the wrapper is broken or not. The work on content-based wrapper verification so far has relied wholly or partly on HTML *token density*, as a sign of a broken wrapper. By the term HTML token density we mean the ratio of the number of HTML tokens (predefined tokens for the HTML language, such as `<b>`, `</i>`, etc.) to the total number of tokens of the extracted token sequence. HTML token density is a useful indicator of a broken wrapper, because HTML tokens appear rarely in the information that one wants to extract, while due to their parsing nature wrappers tend to extract HTML tokens when they are broken. Thus, a high HTML token density is a strong indicator of a potential problem.

Despite the intuitive use of HTML token density as an indicator of potential problems, there are many cases where this simple heuristic does not work, leading usually to false alarms, i.e., cases where the wrapper is not broken, but still extracts HTML tokens. An indicative example of this fact can be demonstrated using

Web page source
`<b>Godfather</b> <i>Scorzese</i> <b>Dogville</b> <i>Lars von Trier</i> <b>Volver</b> <i> Pedro Almodovar </i>`

Rendered Web page
Godfather *Scorzese*
Dogville *Lars von Trier*
Volver *Pedro Almodovar*

Initial wrapper

```
procedure ccwrap1(page $P$)
while there are more occurrences in $P$ of "<b>"
  for each {<$l_k$,$r_k$>} in {<"<b>","</b>">,<"<i>","</i>">}
    scan in $P$ to next occurrence of $l_k$
    save position as start of $k^{th}$ attribute
    scan in $P$ to next occurrence of $r_k$
    save position as end of $k^{th}$ attribute
return extracted {..; <movie title, director> ;..} pairs
```

Extracted pairs `<movie title, director>`
<Godfather, Scorzese>
< Dogville, Lars von Trier>
< Volver, Pedro Almodovar>

Table 1: Example of a wrapper.

table 3. One can imagine a case where the user wants to extract numerical values typically consisting of an integer and a fraction. Previous content-based verification systems that rely significantly on the HTML token density, are likely to produce a false alarm during the testing.

However, the crucial step towards the reduction of false alarms is the introduction of the penalty system. The main reason that prior work suffers from false alarms is the reliance on the syntactic regularities of the extracted data. One can easily imagine several examples of changes in the layout that would raise false alarms using prior work: 320\$ $\rightarrow$ 320USD, 120,000,000 $\rightarrow$ 120M. In other words, prior work has not attempted to capture the *semantics* of the extracted data. This is the main goal of the penalty system introduced in this work.

2.2 Related Work

There are two main approaches to wrapper verification: structure-based and content-based. Structure-based methods rely on the positioning of interesting information, i.e., information that we want to extract, on the DOM structure of the HTML pages. Assuming that the structure remains fixed, unless an important change

Web page source
`Movie:  Godfather, <b>Scorzese</b>`
`Movie:  Dogville, <b>Lars von Trier</b>`
`Movie:  Volver, <b>Pedro Almodovar</b>`
Rendered Web page
Movie: Godfather, **Scorzese**
Movie: Dogville, **Lars von Trier**
Movie: Volver, **Pedro Almodovar**
Erroneously extracted `<movie title, director>` **pairs using *ccwrap1***
$<$ Scorzese, $\emptyset >$
$<$ Lars von Trier, $\emptyset >$
$<$ Pedro Almodovar, $\emptyset >$
Modified wrapper

```
procedure ccwrap2(page $P$)
while there are more occurrences in $P$ of "Movie:"
  for each {<$l_k$,$r_k$>} in {<"Movie:",",">,<"<b>","</b>">}
    scan in $P$ to next occurrence of $l_k$
    save position as start of $k^{th}$ attribute
    scan in $P$ to next occurrence of $r_k$
    save position as end of $k^{th}$ attribute
return extracted {..; <movie title, director> ;..} pairs
```

Table 2: Modification of the web page layout and of the wrapper.

of layout has occurred, they are looking for such changes at regular time intervals [18]. In contrast, content-based methods, such as the one proposed in this paper, examine the content of the extracted information, assuming that it presents regularities. A significant deviation in the type of extracted information signals a potential change in the layout of the page that has resulted in the erroneous extraction of information. In this section we focus on content-based verification approaches.

A common characteristic of the content-based methods is the use of a reference set of extracted information, which is trusted to be correct. This set is often called *training set*, due to the fact that it is used to induce regularities in the extracted information. This training set is constructed during the normal operation of a wrapper that is known to be correct. In order to test whether the wrapper is working properly, at a later point in time a second set of extracted information is generated that is called the *test set*. Figure 3 illustrates this basic content-based approach to wrapper verification, where the verification system uses the training and the test set to produce an indication of whether there is a potential problem with the wrapper or not.

The simplest content-based method is a simple regression tester, called EXACTMATCH [9]. This method

Training Data
83
53
51
57 <sup>4</sup><sub>5</sub>
Test Data
45<sup>2</sup><sub>5¡/sub>
61<sup>3</sup><sub>7¡/sub>
103<sup>1</sup><sub>8¡/sub>
72<sup>1</sup><sub>8¡/sub>

Table 3: A real-world example where relying on HTML token density is likely to lead to a false alarm.

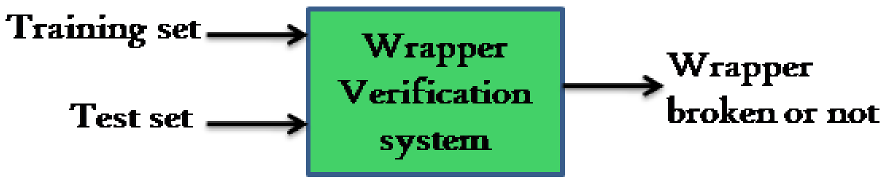

Figure 3: Basic approach to content-based wrapper verification.

assumes that Web sources return always the same results for a fixed query if there is no change in the dynamic Web page generation. Thus, any changes to the response of a source to a query indicates a potential change. This assumption is not valid since most Web sources are dynamic, changing continuously their content and thus their response to query.

The first content-based approach that used machine learning techniques for wrapper verification was RAPTURE [10]. RAPTURE uses a set of global numeric features to characterize the extracted information. These features are word count, average word length and densities of various types of token, such as HTML tokens, punctuation, upper case letters, etc. The algorithm assumes that these features are random variables, following a normal distribution whose mean and variance are estimated from the training set. RAPTURE uses these distributions to estimate the probability that the data extracted from the test set follow different distributions over the feature variables. Based on these, a total verification probability is estimated, which indicates whether the wrapper is correct or not. If this verification probability is greater than a threshold the algorithm decides that the wrapper is correct, otherwise that it is broken. An interesting observation in the experimental results of RAPTURE is that it achieves its best results by using HTML density as the single numeric feature. Adding other numerical features reduces significantly its performance.

A similar set of global numeric features is used in [12]. The main improvement over RAPTURE is achieved through DATAPROG, a syntactic pattern learning algorithm which describes how the extracted data usually begin and end. DATAPROG searches for patterns, sequences of syntactic token types and single tokens that happen to occur more frequently than what we would expect if token types were randomly and independently generated. Initially each token is assigned one or more token types from a token type hierarchy. Then a greedy iterative algorithm is used to learn a set of significant patterns from the token-type data. The learned patterns are encoded in a prefix tree, where each node is either a token type or a token. Every path from the root of the tree to a leaf corresponds to a significant syntactic pattern and is constructed incrementally, starting from the root and finding significant specializations of patterns that do not subsume each other.

By applying DATAPROG to both the training and test set, one obtains a set of significant syntactic patterns for each of the two sets. The number of examples covered by each of these patterns, together with the values of other numeric features, such as those mentioned for RAPTURE, are used to form a training and a test vector. Thereafter, a goodness of fit test (Pearson's chi-square test) is performed to decide if the two vectors are correlated. The patterns are used in this test as follows: The wrapper verification algorithm assumes that a syntactic pattern which is common between the training and test dataset will describe the same proportion of data records in these datasets. Thus, the term $\frac{(t_i - nr_i/N)^2}{nr_i/N}$ is added to the Pearson's statistic for each pattern which is found to be common in the two datasets. Variables t_i and r_i correspond to the observed number of data records expressed by the common pattern in the test and train set respectively, while n and N are the total counts of data records in the two sets. The probability of the estimated value of Pearson's statistic for the given degrees of freedom, i.e., the cardinality of the two vectors, according to the chi-square distribution is compared to the significance level of 0.05. If it is below the threshold, then the response of the system is that the wrapper is broken.

A hybrid approach is presented in [15]. MAVERIC is a combination of both content and structure based approaches since it takes into account both the HTML format of the query results and several similar numerical features with those used in DATAPROG and RAPTURE.

DATAPROG, the state-of-art content-based method can detect syntactic changes. As we described previously, it is a common phenomenon that a slight syntactic change may result in a false alarm, since the semantics of the extracted data remain the same. Another reason that the assumption made by DATAPROG about similar coverage between training and test dataset for the common patterns is limiting is our wish to train our system using a small amount of data, possibly not exhibiting all possible syntactic patterns. On the contrary, VEWRA is designed having in mind the semantics of the extracted data. Our penalty system is a step towards this direction: using the notion of correlated patterns and several other well-intuitioned parameters, we were able to reduce significantly the false alarms, while maintaining the good performance of prior work in all other aspects.

3 Proposed Method

In this section we present the new wrapper verification algorithm, VEWRA, which aims to reduce the number of false alarms while maintaining high accuracy. At a macroscopic level, our approach is comprised of three well-defined steps. This is shown in figure 4. In the following, we describe each step in detail, emphasizing the contribution of our method with respect to past work. Table 4 gives a list of symbols used and their definition.

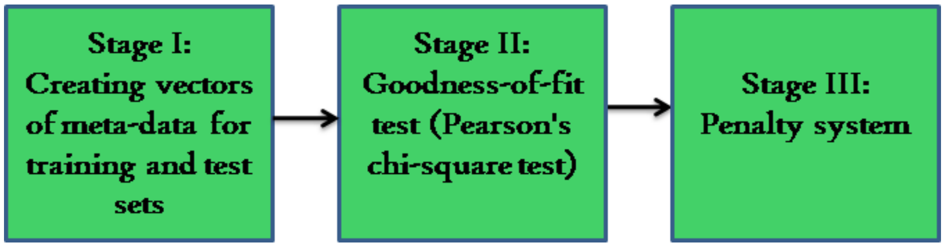
Stage I:
Creating vectors of meta-data for training and test sets
Stage II:
Goodness-of-fit test (Pearson's chi-square test)
Stage III:
Penalty system

Figure 4: The VEWRA wrapper verification process.

Symbol	Definition
R	Training dataset
T	Test dataset
N_R	Number of training data records
N_T	Number of test data records
P_R	Patterns learned by algorithm 1 using R
P_T	Patterns learned by algorithm 1 using T
P'_R	Pruned P_R
P'_T	Pruned P_T
P_C	Common patterns, P_C = $P'_R \bigcap P'_T$
p_u	Pattern appearing during test but not training, $p \in P'_T - P'_R$
n_{CT}	Number of test data records expressed by P_C
$\chi^2_{\nu,\alpha}$	Critical value of chi-square distribution with ν degrees of freedom at α significance level
X	Value of Pearson's χ^2 statistic
TP (true positive)	correctly identified changed sources
TN (true negative)	correctly recognized unchanged sources
FP (false positive)	false alarms
FN (false negative)	unidentified changed sources

Table 4: Definitions of frequently used symbols.

3.1 Step 1: Creating features

Similar to prior content-based approaches, we encode the training and test sets into two feature vectors, which we call the *training* and *test* vector respectively. Each vector is composed of a *numerical* and a *syntactic part*. The numerical part comprises of the standard measures used in existing approaches, with the exception of the HTML token density feature. As explained in section 2, we have chosen not to use this, in order to avoid the false alarms that it tends to generate. Thus, the numerical part of our vectors consist of the following features: a) letter density, b) punctuation density, c) digit density, d) mean word length and e) mean number of tokens per data record.

The syntactic part of the vector takes the form of the prefix tree that was introduced in [12], i.e., a tree is learned, based on the begin patterns of relevant data records. Each node of this tree corresponds to a token type and each path from the root to a leaf to a syntactic pattern that occurs with sufficient statistical significance in the data set.

VEWRA uses the token type hierarchy shown in figure 5. A lexical analyzer tokenizes and characterizes the provided data records, according to this type hierarchy. In contrast to DATAPROG, each token is assigned exactly one token type, namely the most specific one it can take, i.e., the type closest to the leaves of our hierarchy. To illustrate this, the token “CS123” is assigned only the type ALPHANUMERIC, the token “PeertoPeer” the type ALPHABETIC, the token “Greece” the type CAPITALIZED and the token“17.5” the type DECIMAL. In contrast to DATAPROG, only token types and not tokens themselves are used in the constructed patterns.

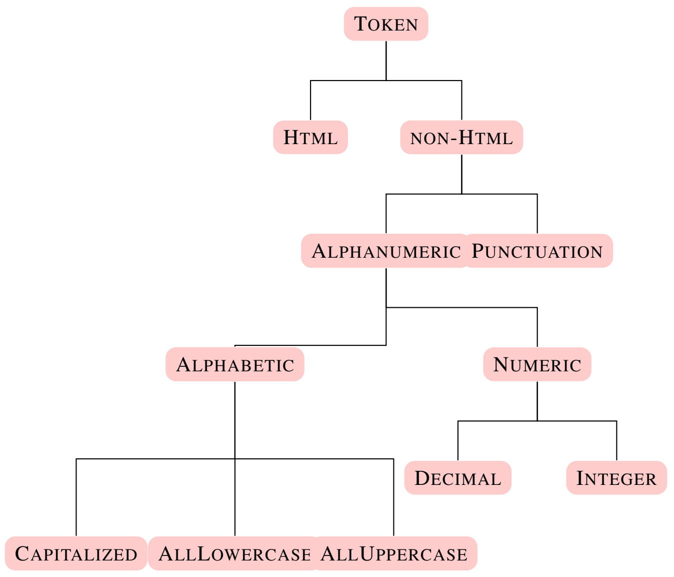

Figure 5: The VEWRA token type hierarchy.

Using the types of Fig. 5, the pattern learning algorithm generates a prefix tree for each extracted data field. The learning algorithm uses variable-length begin patterns. In particular, the number of tokens that are considered to be the beginning of a token sequence is a logarithmic function of the average size of the sequences extracted for a particular field, e.g. `movie title` or `movie director`. The rationale behind this is that different fields differ significantly in the length of the extracted token sequences. Thus, despite the fact that the starting tokens of a sequence are most important for pattern learning, the size of the

Node information	**Description**
N.tokentype	the token type of the node, according to the hierarchy of Fig. 5
N.pattern	the pattern of the node corresponding to the path from the root to this node
N.examples	the set of data records that the pattern of the node covers
N.children	the set of children nodes

Table 5: Description of the information contained in each node in the pattern tree.

patterns should be slightly longer for those fields that are usually longer. VEWRA also makes use of the WILDCARD token, which is used to produce significant patterns even when small differences occur in the middle of otherwise similar token sequences.VEWRA pattern learning algorithm resembles DATAPROG. There are important differences though. To illustrate an important difference between the VEWRA pattern learning algorithm and DATAPROG imagine the following case: we want to find a pattern of length three and in the second position there is no statistical important token type. DATAPROG will terminate producing therefore a pattern of lenght one whereas our algorithm will produce a pattern of lenght three with the WILDCARD token type in the middle. Finally, recall that in contrast to DATAPROG we do not make use of tokens in our patterns but only of token types. Using tokens in syntactic patterns can be misleading and thus lead to false alarms. For instance, if there are many addresses in the training data, but they are all from Pittsburgh, tokens are not the best evidence for addresses, because it would not recognize addresses from Athens.

Algorithm 1 describes the pattern learning process. The algorithm begins by constructing the root node of an empty tree. Each node N in this tree is an object containing the information described in table 5.

The pattern learning algorithm iterates n times, equal to the estimated length of begin patterns for the corresponding data field. In each of the n iterations the tree is extended by adding children nodes that form significant sub-patterns. This is done, by examining each of the current leaves of the tree (set S) and performing a statistical significance test for each token type in Fig. 5, excluding the type HTML.

For the sake of illustration, we will describe one step of the algorithm here. Let us assume that our wrapper extracts information from Web pages that describe laptop computers. In particular, we will look at the extraction of the field `model` of a laptop. Table 6 presents some example token sequences that correspond to this field. First the algorithm determines that the appropriate length of begin patterns for this field is $n = 2$. Thus the prefix tree will be two levels deep. If we further assume that the first level of significant 1-token patterns has been generated and the node M with M.tokentype=CAPITALIZED, is among the children of the root node, we examine which token types can be added as children of this node. Evaluating the token type ALPHANUMERIC as a candidate, we create a new temporary node N, with the following information:

N**.tokentype** : ALPHANUMERIC,

N**.pattern** : "CAPITALIZED ALPHANUMERIC",

N**.examples** : {"Everex NM3500", "Mitsubishi AA141XA01",

Data Records
Sony VAIO Z600LEK
Toshiba Qosmio G35
Toshiba Tecra A8
Everex NM3500
Sony VAIO TXN17PB
HP Pavillion dv6227cl
Acer Aspire 5100
Mitsubishi AA141XA01
Panasonic EDTCB21QAF
IBM ITXG77C
IBM ITXG76E2
IBM ITSX68C
Sony VAIO VGN-FE41E00

Table 6: Example of extracted information for the field `model` from a laptop web site.

"Panasonic EDTCB21QAF"},

N**.count** : 3,

N**.children** : $\emptyset$.

After the creation of the temporary node, we form the null hypothesis that *N*'s pattern is not statistically significant. Null hypothesis testing is performed by the method *significant*(*N*.count, *M*.count, OccurProb(*N*.tokentype)), where in our example *N*.count=3, *M*.cou- nt=9 and OccurProb(*N*.tokentype)=$\frac{1}{8}$, as there are 8 different token types and we have assumed that they are a-priori equiprobable[3]. Assuming that the random variable of the number of occurrences of a token type follows a normal distribution, as an approximation of the binomial, we perform hypothesis testing with a significance level $\alpha = 0.05$ ([7]). In this particular example, the null hypothesis is rejected and thus we add the temporary node N as child of M.

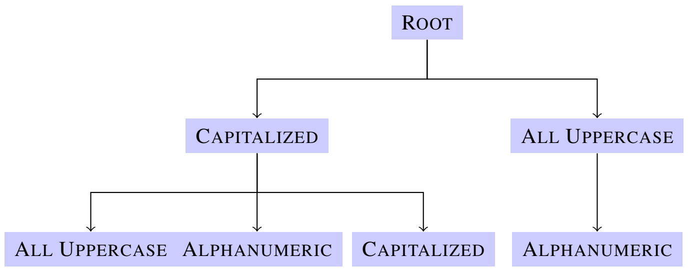

Figure 6: The corresponding pattern tree for the data records in table 6.

This procedure is repeated, adding a node for each token type that is considered significant. If none of the token types leads to statistically significant patterns, we add a single child to M with a *wildcard* token

[3]In general, these prior probabilities could be specified by the user incorporating therefore prior qualitative knowledge about the extracted data (e.g extracted information for the field `stock prices` would imply higher prior probabilities for the numerically related token types).

type. Thus, a wildcard node indicates that there is no significant token type to continue the pattern from the root to its parent. It is worth stressing again that HTML token types are excluded from this process and thus they will never be added to the tree. Figure 6 shows the pattern tree that is created for the above example.

3.2 Step 2: Goodness-of-fit test

In this step, Pearson's chi-square ([7]) is used to test the goodness of fit of the numerical part of the training and testing vectors. If the testing vector frequency distributions differ significantly from the frequency distributions of the training vector, our verification system concludes that the wrapper is broken and does not move to the third step. As already mentioned, HTML token type density is not used in this test.

It is at this point that the most important difference between VEWRA and DATAPROG appears. In [12], the goodness-of-fit test is also applied to the frequency of occurrence of the patterns that DATAPROG discovers. Thus, if several patterns have very different frequency in the two vectors, the system will consider the wrapper as broken. This handling of pattern frequencies is a major cause of the observed false alarms, as it is very often the case that these frequencies are different, without necessarily meaning that the wrapper must change. Therefore, we propose to avoid using the syntactic patterns in the goodness-of-fit-test as the prior work did. We take advantage of the patterns via a penalty system, which forms the third and final step of VEWRA.

3.3 Step 3: Penalty system

If X, the computed value for Pearson's statistic from the previous step is less than the critical value, $\chi^2_{4,0,05}$[4], the verification system enters the third step, which is the penalty system. It is called penalty system, since it increases Pearson's statistic without increasing the degrees of freedom and subsequently the critical value $\chi^2_{4,0,05}$.[5] This is described by the following equation:

$$X \leftarrow X + r{*}\chi^2_{4,0,05}, \text{ where } 0 \leq r \leq 1,$$

Before we explain the penalty system, we present its "big" picture. On the one hand, the penalty system relaxes the strict requirement of prior work for almost exactly same syntactical patterns between the train and test datasets in order to avoid the false alarms. On the other hand, we want to retain the good performance on all other aspects that prior work exhibits. Therefore, the penalty system assumes that if the wrapper is working well on the test data, then the test and training vectors need to share several patterns in common. Thus, the penalty system allows more differences between P_R and P_T than prior work. If these differences are considered to be "fundamental", then the penalty system decides that the wrapper is broken by imposing a severe penalty to X. If however these changes are "small" and can be attributed to insufficient training[6], then the penalty system does not trigger an alarm. The question now is which differences are considered "fundamental" and which "small". This is answered through several new notions introduced in this work, which aim to infer whether the train and test datasets have the same underlying semantics. The quantitative details of the penalty system (all the cases and exact r values) will be presented in a longer version of this paper. Here, we treat the penalty system rather qualitatively, showing with which criteria we distinguish the different cases of the penalty system and how we choose the penalty values.

[4]There are four degrees of freedom since five numerical features in step 2.For α we use the typical value 0,05.

[5]As the degrees of freedom of the chi-square distribution increase so does the critical value $\chi^2_{4,0,05}$, provided that significance level α remains the same.

[6]In the sense that not all patterns appeared during the training procedure.

Table 7 shows synoptically the parameters and the concepts used in the penalty system with the rationale behind them in a “question-like”/motivating way. In the following, we explain them in more detail.

Let us denote with P_R and P_T the training and test pattern sets respectively. We recall here the manner in which the length of these patterns is determined, which takes into account the average length of the extracted data records. Due to this procedure, we expect the length of the patterns in the two vectors to be comparable per extracted field. If this was not the case, the goodness-of-fit test, that compares the average length of the extracted records, would have failed and we would not have entered step 3. Nevertheless, in order to make the two pattern sets more comparable, we prune patterns in one of the sets if they are longer than the patterns in the other set, resulting in the pruned versions of the sets P'_R and P'_T.

The first important concept that we introduce for the purposes of effective pattern comparison is the concept of *correlated pattern sets*. Its goal is to predict cases where the differences between P'_R and P'_T are due to insufficient training, which causes some patterns not to have been encountered in the training dataset and therefore to be excluded from P'_R. The characterization of pattern sets as correlated is based on the determination of *correlated token types*, according to the hierarchy of Fig. 5. Based on this, three *correlation groups* are formed according to table 8, associating different token types.

Token types in the same group are considered “interchangeable”, i.e., it is more probable that in the position of a token type of a specific group we will encounter another token type from the same group than one from another group. The WILDCARD and PUNCTUATION token types are included in all groups.

Based on this definition of the three *correlation groups*, two patterns are considered correlated if the number of correlated token types in a one-to-one correspondence of the patterns exceeds a number which is a non-decreasing function of the number of the token types of the smallest of the two patterns. The philosophy of this function is that short patterns need to match almost exactly, while longer ones are far less likely to match. Therefore, we can tolerate a higher degree of uncorrelated tokens in longer patterns. Pattern correlation is clearly a reflexive relation. Based on this definition of correlated patterns, we define also correlation at the level of the pattern sets, as the requirement for each pattern in each set to be correlated with at least one pattern in the other set.

In order to illustrate the process of correlation, assume two pattern sets P_R={< ALL UPPERCASE, INTEGER>, <CAPITALIZED, INTEGER>} and P_T={<ALL LOWERCASE, INTEGER, ALL UPPERCASE>}. The set with the shortest patterns is P_R, since its patterns are of length two, and P_T's single pattern will be pruned to P'_T={<ALL LOWERCASE, INTEGER>}. Assuming that the correlation requirement for two patterns of length two is to have both tokens correlated, each of the two patterns in P'_R are correlated with the single pattern in P'_T and vice versa. As a result the sets P'_T and P'_R are correlated. This notion of partial pattern matching can prevent many false alarms.

Penalty cases Based on the pruned pattern sets, we calculate the set of common patterns P_C as the intersection of the two sets $P_C = P'_R \bigcap P'_T$. The penalty system considers several different cases based primarily on the cardinalities of these three sets. For small cardinalities (≤ 4) which are more frequent, we have created all possible different cases, whereas for larger cardinalities we have created closed regions of length 3 (all but the last one which is open, i.e $[10, \infty)$).

Penalty values The penalty values also depend on the relative size of P_C. The larger it is, the higher the similarity between the training and test data and thus the lower the penalty should be. In the case where $P'_R = P'_T$, the penalty given to X is zero. At the other end of the spectrum, if $P_C = \emptyset$, then the penalty is the highest. The maximum value of the penalty, as all other penalty values too, differs according to the values which the parameters in table 7 take for the given training and test datasets.

More specifically, the penalty system is based on the following criteria:

- *Relative size of P_R and P_T*: If there are no or very few patterns in P_C, the more patterns in P_R and

Concepts & Parameters	**Rationale**
Correlated patterns	Given the single training pattern CAPITALIZED, why should we penalize the same the testing patterns DECIMAL and ALLUPPERCASE?
Length of patterns in P_R, P_T	For an unseen pattern p_u, why should we give the same penalty when the length of the patterns is one (small potential pattern diversity) and when it is six (large potential pattern diversity)?
N_R	Why should we give the same penalty to an unseen pattern p_u when we have 10 and 10^4 training records respectively?
$\|P'_R\|$,$\|P'_T\|$,$\|P_C\|$	Given that $\|P_C\|$=1, should we give the same penalty when $\|P_R\|$=$\|P_T\|$=10 and $\|P_R\|$=$\|P_T\|$=2?
n_{CT}	Given P_R, P_T, P_C and 10^6 testing records, why should we penalize the same the case of 80% of the testing records satisfying some common pattern with the case of only 50%?

Table 7: Rationale behind the parameters that determine the penalty.

Association Group	**Token Types**
ALPHABETIC	ALPHABETIC, ALL UPPERCASE, ALL LOWERCASE, CAPITALIZED, PUNCTUATION, WILDCARD
NUMERIC	NUMERIC, DECIMAL, INTEGER, PUNCTUATION, WILDCARD
ALPHANUMERIC	ALPHANUMERIC, PUNCTUATION, WILDCARD

Table 8: Correlated token types.

P_T the higher the penalty will be. For example, our system gives a higher penalty when

> P'_R={<ALL UPPERCASE, INTEGER>, <ALL UPPERCASE, CAPITALIZED>, <ALL UPPERCASE, DECIMAL>} and
> P'_T={<DECIMAL, PUNCTUATION>, <WILDCARD, PUNCTUATION>, <INTEGER, ALPHABETIC>},

than when

> P'_R={ <ALPHABETIC> } and P'_T={<CAPITALIZED, INTEGER>}.

In both cases P_C is empty but in the first case, P'_R and P'_T are large and so we can claim to be more certain that it is not a coincidence not having seen any patterns in common.

- *Length of patterns*: Using this parameter we aim to capture the possible pattern diversity. The larger the length is, the more patterns we assume that could possibly appear during the testing and therefore, the smaller penalty we impose to an unseen pattern p_u.
- n_{CT}: The penalty will be low, if the test patterns that do not appear in the training patterns correspond to only a small number of test records. For example, let us consider the case where P'_R,P'_T and P_C have 4, 5, 2 patterns respectively. If the two common patterns describe 40% of the test data records, the penalty is higher than if they describe 50% of the test data.
- N_R: The more data records we have seen during, the more certain we are that we have encountered most of the existing patterns. Thus, when we encounter a low degree of similarity of the training and test data, we are more confident that something is wrong with the wrapper and thus the penalty is higher. To be more precise, we defined the number of training data records required to be sure that we have seen almost all patterns proportional to k^m, where m is the number of starting tokens and k is a constant[7].

The above criteria aim to predict when there is high possibility that a pattern has the same underlying semantics with another. When correlation is high, the penalty is low, despite the fact that the patterns do not match exactly.

[7] k was set to 4, the cardinality of the largest group of associated token types,excluding PUNCTUATION and WILDCARD. This is in accordance with the assumption that only correlated patterns can appear and with the miltiplicative counting principle.

Measure	Definition
Accuracy A	$A = \frac{TP+TN}{TP+TN+FP+FN}$
Changed precision P_c	$P_c = \frac{TP}{TP+FP}$
Changed recall R_c	$R_c = \frac{TP}{FN+TP}$
Changed F-measure F_c	$F_c = \frac{2P_cR_c}{P_c+R_c}$
Unchanged precision P_u	$P_u = \frac{TN}{TN+FN}$
Unchanged recall R_u	$R_u = \frac{TN}{TN+FP}$
Unchanged F-measure F_u	$F_u = \frac{2P_uR_u}{P_u+R_u}$
Overall F-measure F	$F = \frac{2F_cF_u}{F_c+F_u}$

Table 9: Measures commonly used to evaluate the performance of a wrapper verification system.

The value of the parameter r was set for each of the cases using 10-fold cross validation. The values tested for r were all values from 0 to 1 using a 0,05 step. Since ties appeared for close values of r, we resolved them randomly. For the cases that never appeared in the available data (for example pattern lengths greater than 10) we set the r values by our intuition. The system has proved to be insensitive to small perturbations of the cross-validated r values. These values yield in very good results presented in section 4.

Penalty system's decision If by the end of the penalty system Pearson's statistic X remains smaller than $\chi^2_{\nu,\alpha}$, the system considers the wrapper to extract information correctly. In the opposite case, the system suggests that the wrapper is broken.

4 Experimental Results

We evaluated the VEWRA algorithm on the two datasets used in [10] and [12], which we call the RAPTURE and DATAPROG datasets respectively. For the evaluation we use the widely used measures in Information Retrieval. These metrics are shown in 9.

In the following two subsections we present the results of VEWRA on the two datasets. STALKER ([16]) was used for wrapper induction, but it could easily be replaced by any other method.

4.1 Experiments with the RAPTURE dataset

4.1.1 Set-up of the experiment

For each Web source in the dataset, we are provided with a sequence of Web pages sampled at different time points. Figure 7 provides an example of this process, where two changes to the Web source have occurred at time points t_2 and t_4. The capital letters below the time line denote subsets of the data sampled over a subperiod, e.g. subset A is sampled over the period $[t_0..t_1)$. Each such subset typically contains a very small number of Web pages.

Based on this sequence, a number of experiments are produced, each associated with a corresponding triple (X, Y, Z) of subsets of the data. The first of the three subsets X is used to train the wrapper, which is then applied to the second Y and the third Z subset to produce the training and test vector respectively for VEWRA. For instance, the experiment (A, B, C) indicates that a wrapper is produced using subset A of Fig. 7 and VEWRA is asked to compare the information extracted with this wrapper from subsets B and C. Since the layout of the source has changed at t_2, the result of the verification should be *Changed*.

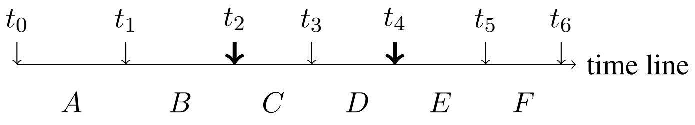

Figure 7: An example of a sequence of Web pages sampled from a data source.

Using this notation, the set of experiments and the expected outcome of VeWra corresponding to the example of Fig. 7 is the following:

- (A, A, B): Unchanged, $(A, A, C \bigcup D)$: Changed, $(A, B, C \bigcup D)$: Changed
- (C, C, D): Unchanged, $(C, C, E \bigcup F)$: Changed, $(C, D, E \bigcup F)$: Changed
- (E, E, F): Unchanged

In this manner, we test VeWra on a variety of cases, including some cases where the training data of the wrapper are used to produce the training vector for VeWra, as well as cases where both the training and the test vector are produced from data that were not used when training the wrapper.

4.1.2 Results

According to the above set-up and using 26 Web sources[8] from the Rapture dataset, we generated 1713 experiments for VeWra. In this set of experiments, VeWra committed only 4 mistakes and these were of the type FP, i.e., false alarms. This is the least severe type of error, as the wrapper induction system will be asked to re-induce the correct wrapper. Using the measures introduced at the beginning of the section, we have obtained the results shown in table 10. The results for Rapture in table 10 are the ones reported for the best possible performance in [10]. We present our performance using the unreported measures as well.

Out of the four FP mistakes, two of them occurred in experiments from the `altavista` source and the other two from the `irti` source. In the former case, the test dataset contained data records with many more punctuation marks than the training dataset. As a result the mean number of tokens and the punctuation density was highly increased in the test vector. The system did not enter the penalty system because Pearson's statistic X was greater than the critical value $\chi^2_{\nu,\alpha}$. Thus, by the end of the second step our wrapper verification system considered that the wrapper was broken. In the `irti` experiments, the test data records contained many more tokens per record than in the training data. Thus, X exceeded again the critical value, when the term that compares the mean number of tokens per record in the two datasets was added.

Regarding the evaluation of the penalty system itself, no errors were committed, i.e., the value of X was always increased above the threshold value $\chi^2_{\nu,\alpha}$, when a change should be detected. It is worth noting that in more than half of these cases, the result of the second step, i.e., the goodness-of-fit test on the global numeric features, did not indicate that the wrapper was broken, but the use of the penalty system led to the correct result. At the same time, the value of the Pearson's statistic was maintained below the threshold, in all cases in which no change had occurred. Thus, the penalty system exhibited excellent performance. The results shown in table 10 for Rapture are the ones reported when using only Html token density.

[8]The dataset contained 27 sources but our wrapper induction system did not generate a usable wrapper for the `rain` source.

	VEWRA	RAPTURE
A	0.998	0.9999
P_c	0.991	-
R_c	1.000	-
F_c	0.996	0.9999
P_u	1.000	-
R_u	0.997	-
F_u	0.998	0.93
F	0.997	0.96

Table 10: Performance of VEWRA and RAPTURE.

We observe that there is not a clear "knockout". Notice also that according to [10], if any other numerical feature except the HTML token density -which as explained can be a source of false alarms- is used, the performance drops significantly.

4.2 Experiments with the DATAPROG dataset

4.2.1 Set-up of the experiment

The main difference of this from the RAPTURE dataset was that we did not need to generate any wrappers, as the data is provided in the form of extracted tuples. In fact, using the example of Fig. 7, the tuples provided for subsets A, B, D, F are extracted correctly, while these provided for the subsets just after the change, i.e., subsets C, E, are incorrect, as they are extracted by the "old" wrapper. Based on this observation, and removing the wrapper induction subset from the triple-notation used above, we generate the following set of experiments for the data of Fig. 7:

- (A, B): Unchanged, (B, C): Changed, (B, D): Unchanged
- (D, E): Changed, (D, F): Changed

In other words, we generate experiments with correctly extracted data before and after a change, expecting VEWRA to respond with *Unchanged.* Furthermore, since the data are provided already separated in chronologically arranged samples, more than two samples may be provided between two change points. In this case, only the first subset contains erroneous data, while the rest are extracted by the modified wrapper, e.g. in the example of Fig. 7, subset D may be split further into subsets D_1, D_2, etc., all correctly extracted. In this case, we generate the corresponding *Unchanged* experiments: (B, D_1), (D_1, D_2), etc.

4.2.2 Results

According to the above set-up and using 17 Web sources[9] from the DATAPROG dataset, we generated 184 experiments for VEWRA. In this set of experiments, VEWRA committed only 3 mistakes and these were of the least severe FP type, i.e., false alarms. Using the same measures as above, we have obtained the results shown in table 11(see also fig. 1).

Once again, the results are very encouraging, even more so due to the fact that the penalty system was not responsible for any of the three errors. Two of the errors were due to a high increase in the frequency

[9]Unfortunately only a subset of the data were available and thus we were only able to conduct a subset of the experiments reported in [12].

	VEWRA	DATAPROG
A	0.984	0.951
P_c	0.857	0.680
R_c	1.0	0.944
F_c	0.923	0.791
P_u	1.0	0.994
R_u	0.982	0.952
F_u	0.991	0.972
F	0.956	0.872

Table 11: Performance of VEWRA and DATAPROG.

of punctuation marks in the test dataset of the Web sources `borders` and `whitepages`. This has led to a substantial increase in the mean number of tokens per data record and the punctuation density. The third false positive was due to a high reduction in the letter density in the test dataset of the source `quote`.

On the 184 experiments we conducted, the DATAPROG system committed 9 errors, of which 8 FP and 1 more serious FN, which means that a change went unnoticed. The advantages of the proposed method are apparent from the table 11: the number of false alarms is reduced drastically, without introducing any costly false negatives.

5 Conclusions

The main contribution of this paper is VEWRA, a novel content based algorithm for wrapper verification. The novelty of our method lies in the fact that it captures not only the syntactic patterns as its predecessors, but also the correlations among them. Therefore, despite the simple underlying ideas of VEWRA, our method compared to the prior work is more robust and its performance is almost ideal. The key ideas of our work are the following: a) an intuitive penalty system which aims to compare the semantics of the training and test information to decide whether the wrapper is broken or not. b) The notion of groups of associated token types and patterns, which helps VEWRA to reduce significantly the false alarms and c) ignore the density of HTML tokens in the information extracted by the wrappers, since it can be a potential source of false alarms too.

The penalty system, introduced by VEWRA is an interesting approach to the problem of wrapper verification. From a higher point of view, the penalty system adopts a bayesian-like approach to the problem of wrapper verification in the following sense: using the notion of correlated patterns (“prior”) and the patterns P_R found to be significant during the training procedure (“likelihood”), it computes a “posterior” over the unseen patterns p_u’s. We think that this deserves further study, due to the clear intuition behind it and to the good results that we obtained. Future work includes adopting this idea to a wrapper re-induction system and also more experiments using data recently crawled to validate further the value of our approach.

Acknowledgements

We would like to thank Nicholas Kushmerick and Kristina Lerman for providing us with the datasets they used.

References

[1] Grigoris Antoniou and Frank van Harmelen. *A Semantic Web Primer, 2nd Edition (Cooperative Information Systems)*. The MIT Press, 2008.

[2] P. Atzeni, A. Masci, P. Merialdo, G. Sindoni, and Universit'a Di Roma Tre. The araneus web-base management system g. mecca.

[3] R. J. Bayardo, Jr., W. Bohrer, R. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, and D. Woelk. InfoSleuth: Agent-based semantic integration of information in open and dynamic environments. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, volume 26,2, pages 195–206, New York, 13–15 1997. ACM Press.

[4] Dayne Freitag and Nicholas Kushmerick. Boosted wrapper induction. In *AAAI/IAAI*, pages 577–583, 2000.

[5] Hector Garcia-Molina, Yannis Papakonstantinou, Dallan Quass, Anand Rajaraman, Yehoshua Sagiv, Jeffrey D. Ullman, Vasilis Vassalos, and Jennifer Widom. The TSIMMIS approach to mediation: Data models and languages. *Journal of Intelligent Information Systems*, 8(2):117–132, 1997.

[6] Michael R. Genesereth, Arthur M. Keller, and Oliver M. Duschka. Infomaster: An information integration system. In *in proceedings of 1997 ACM SIGMOD Conference*, pages 539–542, 1997.

[7] Roger L. Berger George Casella. *Statistical Inference*. Duxbury Advanced Series, 2001.

[8] Craig A. Knoblock, Steven Minton, José Luis Ambite, Naveen Ashish, Pragnesh Jay Modi, Ion Muslea, Andrew G. Philpot, and Sheila Tejada. Modeling web sources for information integration. In *AAAI '98/IAAI '98: Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, pages 211–218, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence.

[9] Nicholas Kushmerick. Regression testing for wrapper maintenance. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI) and Eleventh Conference on Innovative Applications of Artificial Intelligence (IAAI), July 18-22, 1999, Orlando, Florida, USA. AAAI Press / The MIT Press*, pages 74–79, 1999.

[10] Nicholas Kushmerick. Wrapper verification. *World Wide Web*, 3(2):79–94, 2000.

[11] Nickolas Kushmerick, Daniel S. Weld, and Robert B. Doorenbos. Wrapper induction for information extraction. In *Intl. Joint Conference on Artificial Intelligence (IJCAI)*, pages 729–737, 1997.

[12] Kristina Lerman, Steven Minton, and Craig A. Knoblock. Wrapper maintenance: A machine learning approach. *Journal of Artificial Intelligence Research (JAIR)*, 18:149–181, 2003.

[13] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of the Twenty-second International Conference on Very Large Databases*, pages 251–262, Bombay, India, 1996. VLDB Endowment, Saratoga, Calif.

[14] Bing Liu. *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data (Data-Centric Systems and Applications)*. Springer, January 2007.

[15] Robert McCann, Bedoor AlShebli, Quoc Le, Hoa Nguyen, Long Vu, and AnHai Doan. Mapping maintenance for data integration systems. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 1018–1029. VLDB Endowment, 2005.

[16] Ion Muslea, Steven Minton, and Craig A. Knoblock. A hierarchical approach to wrapper induction. In *Proceedings of the Third Annual Conference on Autonomous Agents (AGENTS), May 1-5, 1999, Seattle, WA, USA. ACM*, pages 190–197, 1999.

[17] Ion Muslea, Steven Minton, and Craig A. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems*, 4(1/2):93–114, 2001.

[18] Enghuan Pek, Xue Li, and Yaozong Liu. Web wrapper validation. In *Web Technologies and Applications, Proceedings of the Fifth Asian-Pacific Web Conference (APWeb), April 23-25, 2002, Xian, China.*, pages 388–393, 2003.

[19] Arnaud Sahuguet and Fabien Azavant. Web ecology: Recycling html pages as xml documents using w4f. In *WebDB (Informal Proceedings)*, pages 31–36, 1999.

Algorithm 1 The VEWRA pattern learning algorithm

```
Require: A set of data records $A$
Require: A set of valid token types $TT$
  $\mu \leftarrow$ mean number of tokens per data record
  $n \leftarrow log_2(\mu) + 1$ {length of begin patterns}
  root.tokentype $\leftarrow$ WILDCARD
  root.examples $\leftarrow A$
  root.count $\leftarrow |A|$
  root.pattern $\leftarrow$ "WILDCARD"
  root.children $\leftarrow \emptyset$
  for $i = 1$ to $n$ do
    $S \leftarrow$ GetNodesOfTreeInDepth(root,$i - 1$) {depth 0 corresponds to the root}
    for $j = 1$ to $|S|$ do
      Significant-exists $\leftarrow$ false
      for $k = 1$ to $|TT|$ do
        {New node $N_k$}
        $N_k$.tokentype $\leftarrow TT_k$
        $N_k$.examples $\leftarrow s_j$.examples followed by tokens of type $TT_k$ {$s_j \in S$}
        $N_k$.count $\leftarrow |N_k$.examples$|$
        $N_k$.pattern $\leftarrow$ concat($s_j$.pattern,$TT_k$)
        $N_k$.children $\leftarrow \emptyset$
        if significant($N_k$.count,$s_j$.count,OccurProb($TT_k$)) then
          $s_j$.children $\leftarrow s_j$.children $\bigcup \{N_k\}$
          Significant-exists $\leftarrow$ true
        end if
      end for
      if Significant-exists=false then
        {New node $N$}
        $N$.tokentype $\leftarrow$ WILDCARD
        $N$.examples $\leftarrow s_j$.examples
        $N$.count $\leftarrow |N$.examples$|$
        $N$.pattern $\leftarrow$ concat($s_j$.pattern,WILDCARD)
        $N$.children $\leftarrow \emptyset$
        $s_j$.children $\leftarrow s_j$.children $\bigcup \{N\}$
      end if
    end for
  end for
  return root
```

ML

MACHINE LEARNING
DEPARTMENT

Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213

Carnegie Mellon.