Design of Automatically Adaptable Web Wrappers

Emilio Ferrara\(^1\)  R. Baumgartner\(^2\)

\(^1\)Department of Mathematics  
University of Messina, Italy  
emilio.ferrara@unime.it

\(^2\)Vienna Technische Universität  
&  
Lixto Software GmbH  
Vienna, Austria  
robert.baumgartner@lixto.com

The Third International Conference on Agents and Artificial Intelligence  
ICAART 2011
Outline

1 Motivation
   - Statements and Objectives
   - The Basic Problem
   - Previous Work

2 Our Results/Contribution
   - Tree Matching Algorithms
   - Automatically Adaptable Wrappers
   - Repairable Web Wrapper Agents

3 Future Issues
Outline

1 Motivation
   • Statements and Objectives
     • The Basic Problem
     • Previous Work

2 Our Results/Contribution
   • Tree Matching Algorithms
   • Automatically Adaptable Wrappers
   • Repairable Web Wrapper Agents

3 Future Issues
Statements and Objectives

The actual panorama

- Procedures for extracting data from Web pages exist
- The structure of Web pages changes causing malfunctioning in these procedures
- The manual maintenance of these systems has a cost
- We want to provide an automatic solution to this problem

Objectives of this work

- Introducing a new algorithm optimized for the domain of the problem
- Designing automatically repairable procedures of Web data extraction
- Combining these two techniques to build robust Web Mining solutions
1 Motivation
- Statements and Objectives
- The Basic Problem
- Previous Work

2 Our Results/Contribution
- Tree Matching Algorithms
- Automatically Adaptable Wrappers
- Repairable Web Wrapper Agents

3 Future Issues
Web Data Extraction System

A platform which extracts, automatically and in an 'intelligent' way, data from Web sources (e.g. unstructured Web pages) and stores these data in a structured format in order to make it possible for other applications to use these data.

Wrapper

A procedure (e.g. a logic-based algorithm) which identifies and extracts information (in our case within a Web page). Its functioning is strictly interconnected with the structure of the data source.

Web Data Extraction Systems based on Wrappers

We focus our attention on systems implementing Web wrappers for extracting data from Web pages.
Wrappers - Life Cycle

1. Generation
   - Manual
   - Semi-Automatic
   - Automatic
   - Induction

2. Execution
   - Manual
   - Semi-Automatic
   - By Scheduling

3. Maintenance
   - Manual
   - Automatic and By Adaptation
Wrapper Maintenance

The structure of Web pages changes frequently and without any forewarning. Web wrappers could stop working correctly if the underlying structure changes, also slightly.

Automatic Wrapper Adaptation

Our approach automatizes the process of maintenance of wrappers with an excellent degree of reliability.

Clustered Tree Matching

The algorithm devised for finding similarities/differences between the structure of two Web pages, in order to identify modifications.
The basic problem (1/3): Concepts

HTML Web pages

Web pages are represented adopting a tree data structure, which nodes are elements constituting the page itself.

XPath

The W3C defined this standard language to identify and select elements in the hierarchy of Web pages. From now we assume our Web wrappers relying on XPath(s).

Figure: Example of XPaths selecting one (A) or multiple (B) elements.
The basic problem (2/3): Motivations

- Web pages have rich and complex structures
- These structures change frequently
- Often modifications are invisible (outwardly)
- Modifications happen without forewarning or notifications
- Statistically, small changes are much more frequent than radical

Our Idea

We guessed it was possible to automatize the resolution of this problem through artificial intelligent techniques!
The basic problem (3/3): Pros-Cons

Pros:
- We enhance Web wrappers robustness, thus:
  - Quality of data increases
  - We reduce the risk of malfunctioning of the system
- We reduce the process of manual maintenance, thus:
  - We lower the maintenance cost
  - We allow the staff working on writing new Web wrappers instead of fixing the broken ones

Cons:
- A computational overhead
- The automatic process must be robust in order to minimize malfunctioning
Outline

1 Motivation
   - Statements and Objectives
   - The Basic Problem
   - Previous Work

2 Our Results/Contribution
   - Tree Matching Algorithms
   - Automatically Adaptable Wrappers
   - Repairable Web Wrapper Agents

3 Future Issues
Previous Work

- Tree edit distance and related problems
  - Tree to tree **editing** problem (Selkow, 1977)
  - Tree to tree **correction** problem (Tai, 1979) (NP-hard)

- Web data extraction systems
  - Web data extraction tools and **taxonomical classification** of Web Mining problems (Leander et al. 2002)
  - Lixto Suite: Web data extraction for Web Intelligence and Web Mining (Baumgartner et al., 2009)

- Wrapper maintenance and adaptation
  - **Maintenance** related problems
    (Lerman et al., 2003; Meng et al., 2003)
  - **Wrapper adaptation**, semi-automatic and automatic
    (Wong, 2004; Raposo et al. 2005)
# Outline

1. **Motivation**
   - Statements and Objectives
   - The Basic Problem
   - Previous Work

2. **Our Results/Contribution**
   - Tree Matching Algorithms
   - Automatically Adaptable Wrappers
   - Repairable Web Wrapper Agents

3. Future Issues
Key aspects of STM (Selkow, 1977):

- **Dynamic programming**
- **Recursive** approach
- **Optimal** cost $O(n^2)$ ($n$ is the number of nodes of the biggest between the two compared trees)
- Two small matrices store mapping values, step-by-step, mapping values

---

**Algorithm 2 SimpleTreeMatching($T', T''$)**

1. if $T'$ has the same label of $T''$ then
2. $m \leftarrow d(T')$
3. $n \leftarrow d(T'')$
4. for $i = 0$ to $m$ do
5. $M[i][0] \leftarrow 0$
6. for $j = 0$ to $n$ do
7. $M[0][j] \leftarrow 0$
8. for all $i$ such that $1 \leq i \leq m$ do
9. for all $j$ such that $1 \leq j \leq n$ do
10. $M[i][j] \leftarrow \max(M[i][j - 1], M[i - 1][j], M[i - 1][j - 1] + W[i][j])$ where $W[i][j] = \text{SimpleTreeMatching}(T'(i - 1), T''(j - 1))$
11. return $M[m][n] + 1$
12. else
13. return $0$
Tree Matching Algorithms (2/6): Clustered Tree Matching

Key aspects of our CTM:

- **Assigns weights** to evaluate the importance of a matching.
- **Different behavior** adopted comparing leaves and middle-level nodes.
- Allows a “degree of accuracy” (through a similarity threshold).
- **Identifies clusters** of similar sub-trees.

Algorithm 1: ClusteredTreeMatching($T'$, $T''$)

```python
1: if $T'$ has the same label of $T''$ then
2:   $m \leftarrow d(T')$
3:   $n \leftarrow d(T'')$
4:   for $i = 0$ to $m$ do
5:     $M[i][0] \leftarrow 0$
6:   for $j = 0$ to $n$ do
7:     $M[0][j] \leftarrow 0$
8:   for all $i$ such that $1 \leq i \leq m$ do
9:     for all $j$ such that $1 \leq j \leq n$ do
10:    $M[i][j] \leftarrow \max(M[i][j-1], M[i-1][j], M[i-1][j-1] + W[i][j])$ where $W[i][j] = \text{ClusteredTreeMatching}(T'(i-1), T''(j-1))$
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
```

Figure: A and B are two similar labeled rooted trees. CTM assigns weights to matched nodes. Example: the weight of the node f (in A) is $\frac{1}{2}$ because it appears in B in a sublevel with two children.
Figure: $W$ (weights) and $M$ (matches) pairs of matrices, step-by-step. CTM computes the similarity between these two trees in 6 recursions. The final value of similarity of the two trees is $\frac{3}{8}$. Grey cells identify weights of elements matched between the specific compared subtrees.
Common characteristics of Web pages:
- Rich sub-levels usually represent list items, table rows, etc. (i.e. data)
- Simple sub-levels represents the structure of the page.

Moreover, common modifications are:
- Small changes and happen in deep sub-levels
- Consisting in adding/removing nodes/branches, etc. (i.e. adding/removing details to elements)

Observation
- The algorithm STM ignores these aspects
- The CTM exploits information about the position of matching/mismatching to produce better results
Tree Matching Algorithms (6/6): Advantages and Limitations

Advantages:

- CTM produces an **intrinsic measure of similarity** in the interval \([0,1]\) (while STM returns the distance between two trees)
- A custom degree of **accuracy** can be established introducing a similarity threshold for matching two elements
- The more the structure of compared trees is complex and similar, the better the measure of similarity is accurate (CTM)

Limitations:

- Both approaches can not handle **permutations** of nodes very well
- Both do not work well if new sub-levels of nodes are added/removed

Further considerations

Free text can be matched using classic string matching techniques (string-edit-distance, Jaro-Winkler, bigrams, etc.)
Outline

1  Motivation
   • Statements and Objectives
   • The Basic Problem
   • Previous Work

2  Our Results/Contribution
   • Tree Matching Algorithms
   • Automatically Adaptable Wrappers
   • Repairable Web Wrapper Agents

3  Future Issues
Requirements of the system:

- To store a copy of the structure of the original Web page, using the tree-grammar for simplifying the representation.
- If the wrapper fails, the system compares the structure of the original Web page against the actual one, identifying modifications.

Comparing elements:

- **Nodes**: representing HTML elements, identified by tags.
- **Attributes**: also attributes can be additionally compared.
Automatically Adaptable Wrappers (2/3): Configuration

For configuring the process of automatic adaptation we define:

- A degree of similarity to validate a matching between two elements
- The priority/order of adoption of algorithms (CTM, STM, etc.)
- Additional attributes to be eventually compared
- Integrity constraints on extracted data

Common integrity constraints are:

- Restrictions on the number of occurrences of a specific element
- Data types (e.g. text, values, currency, etc.)
Automatically Adaptable Wrappers (3/3): Example

Figure: An example of Web wrapper adaptation. In the upper area the original page and the original wrapper. In the bottom area the new page and the automatically adapted wrapper, by using the CTM.
Outline

1 Motivation
   - Statements and Objectives
   - The Basic Problem
   - Previous Work

2 Our Results/Contribution
   - Tree Matching Algorithms
   - Automatically Adaptable Wrappers
   - Repairable Web Wrapper Agents

3 Future Issues
Repairable Web Wrapper Agents (1/5)

Wrapper Creation

- Define Output Data Model
- Create Pattern visually
- Configure Adaptation Integrity Constraints
- Store Pattern and Tree-gram Examples
- Hierarchical Pattern Generation

Wrapper Execution and Adaptation

- Deep Web Navigation Steps
- Previous Pattern Eval. Steps
- Evaluation of Pattern
  - Constraints Violated?
  - Yes: Apply Adaptation Logic
  - No: Re-Evaluation of Pattern
  - No: Evaluation of Descendants
  - Yes: Bubble up: re-evaluate ancestor patterns

- Trigger Process Flow Adaptation
- Trigger Setting
- Trigger?
  - Yes: Provide additional information about adaptation during descendant evaluations
  - No

Adaptation Inputs/Outputs

- Stored Example Tree-Grams
- DOM Tree of Current Page
- Current Pattern Configuration (XPath etc.)
- Adaptation Configuration Settings
- Wrapper Adaptation
- Sufficiently similar HTML Subtrees
- Modified XPath Statements
- New Tree-Grams can be stored additionally

Figure: Diagram of wrappers design, execution and adaptation in Lixto VD
Repairable Web Wrapper Agents (2/5)

Image: Lixto VD GUI

- Embedded Browser
- Record/Replay
- Project Repository
- Wrapper Structure
- DOM Tree Structure
- Action Creation and Manipulation
- Network Traffic

Figure: Lixto VD GUI
Repairable Web Wrapper Agents (3/5)

- Each wrapper represents an automatically repairable agent of data extraction
- Several wrappers run in the same environment
- If a wrapper fails, it automatically adapts itself to changes
- Results are collected in a transparent way w.r.t. users

Figure: Architecture of the Lixto Web data extraction platform
**Repairable Web Wrapper Agents (4/5): Experimental Results**

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Use-Case</th>
<th>Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>Social bookmarking</td>
<td>Delicious</td>
<td>40%</td>
</tr>
<tr>
<td>Retail market</td>
<td>Ebay</td>
<td>85%</td>
</tr>
<tr>
<td>Social networks</td>
<td>Facebook</td>
<td>65%</td>
</tr>
<tr>
<td>News</td>
<td>Google news</td>
<td>90%</td>
</tr>
<tr>
<td>Web search</td>
<td>Google search</td>
<td>80%</td>
</tr>
<tr>
<td>Comparison shopping</td>
<td>Kelkoo</td>
<td>40%</td>
</tr>
<tr>
<td>Web communities</td>
<td>Techcrunch</td>
<td>85%</td>
</tr>
</tbody>
</table>

Experimental results:

- **Simple Tree Matching:**
  - The classic algorithm ensures good performances

- **Clustered Tree Matching:**
  - Excellent performances in the domain of Web sources
  - High reliability! \( F_1 - \text{measure} \approx 98\%; \ F_{0.5} - \text{measure} \approx 99\% \)
**Repairable Web Wrapper Agents (5/5): Experimental Results**

![Experimental results](image)

<table>
<thead>
<tr>
<th>Scenario</th>
<th>thresh.</th>
<th>tp</th>
<th>fp</th>
<th>fn</th>
<th>tp</th>
<th>fp</th>
<th>fn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delicious</td>
<td>40%</td>
<td>100</td>
<td>4</td>
<td>-</td>
<td>100</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Ebay</td>
<td>85%</td>
<td>200</td>
<td>12</td>
<td>-</td>
<td>196</td>
<td>-</td>
<td>4</td>
</tr>
<tr>
<td>Facebook</td>
<td>65%</td>
<td>240</td>
<td>72</td>
<td>-</td>
<td>240</td>
<td>12</td>
<td>-</td>
</tr>
<tr>
<td>Google news</td>
<td>90%</td>
<td>604</td>
<td>-</td>
<td>52</td>
<td>644</td>
<td>-</td>
<td>12</td>
</tr>
<tr>
<td>Google.com</td>
<td>80%</td>
<td>100</td>
<td>-</td>
<td>60</td>
<td>136</td>
<td>-</td>
<td>24</td>
</tr>
<tr>
<td>Kelkoo</td>
<td>40%</td>
<td>60</td>
<td>4</td>
<td>-</td>
<td>58</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td>Techcrunch</td>
<td>85%</td>
<td>52</td>
<td>-</td>
<td>28</td>
<td>80</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Total</td>
<td>-</td>
<td>1356</td>
<td>92</td>
<td>140</td>
<td>1454</td>
<td>12</td>
<td>42</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
<th>Simple T. M.</th>
<th>Clusted T. M.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recall</td>
<td>90.64%</td>
<td>97.19%</td>
</tr>
<tr>
<td>Precision</td>
<td>93.65%</td>
<td>99.18%</td>
</tr>
<tr>
<td>F-Measure</td>
<td>92.13%</td>
<td>98.18%</td>
</tr>
</tbody>
</table>

**Figure:** Experimental results
Future Issues

- **Bigrams**: might work well with permutations of groups of nodes
- **Jaro-Winkler**: could better reflect added/missing node levels
- Machine-learning or Natural Language Processing for free text
- **Tree-grammar**: could be used to classify topologies of templates shown by Web pages and to define some *standard execution flow* of extraction
- Spidering techniques: executing tree-grammar templates for harvesting through standard execution flows
Summary

- It is possible to reduce maintenance costs of Web wrappers using techniques of automatic adaptation for facing malfunctioning caused by structural modifications of the Web pages.

- The devised clustered tree matching algorithm ensures excellent robustness of Web wrappers relying on it for the adaptation.

- We combine AI & Agents for obtaining robust Web Mining solutions.

Outlook

- To improve performances facing permutations on nodes.
- To exploit new algorithms and similarity metrics.
For Further Reading I

S. Selkow.
The tree-to-tree editing problem.

K. Tai.
The tree-to-tree correction problem.

A. Leander et al.
A brief survey of Web data extraction tools.

R. Baumgartner et al.
Scalable Web data extraction for online market intelligence.

K. Lerman et al.
Wrapper maintenance: a machine learning approach.

X. Meng et al.
Schema-guided wrapper maintenance for Web-data extraction.

T. Wong.
A probabilistic approach for adapting information extraction wrappers and discovering new attributes.

J. Raposo et al.
Automatic wrapper maintenance for semi-structured Web sources using results from previous queries.