HTML Pattern Generator - Automatic Data Extraction from Web Pages

Mirel Cosulschi¹, Adrian Giurca², Bogdan Udrescu¹, Nicolae Constantinescu¹ and Mihai Gabroveanu¹  
¹Dept. of Computer Science, University of Craiova, Romania  
mirelc, nikyc, mihaiug@central.ucv.ro, udrescu_bogdan@yahoo.com  
²Dept. of Internet Technology, Brandenburg Technical University Cottbus, Germany  
Giurca@tu-cottbus.de

Abstract

Existing methods of information extraction from HTML documents include manual approach, supervised learning and automatic techniques. The manual method has high precision and recall values but it is difficult to apply it for large number of pages. Supervised learning involves human interaction to create positive and negative samples. Automatic techniques benefit from less human effort but they are not highly reliable regarding the information retrieved.

1. Introduction

The semantic content structure of web pages is the principal element exploited by many web applications: one of the latest directions is the construction of wrappers in order to structure web data using regular languages and database techniques.

[Laender et al., 2002] state the problem of generating a wrapper for Web data extraction as follows:

Given a web page \( S \) containing a set of input objects, determine a mapping \( W \) that populates a data repository \( R \) with the objects in \( S \). The mapping \( W \) must also be capable of recognizing and extracting data from other page similar with \( S \).

Any possible solution to the above problem must take into consideration at least two contexts:

First, a HTML page may contain many types of information presented in different forms, such as text, image or Java applets. HTML is a language designed for data presentation, and was not intended as a mean of structuring information and easing the process of structured data extraction. Another problem of HTML pages is related to their bad construction, language standards frequently being broken (i.e. improper closed tags, wrong nested tags, bad parameters and incorrect parameter values).

Second, the web pages from commercial web sites are usually generated dynamically using different server side technologies (JSP, PHP) with data stored in a back-end DBMS. The visitor can easily notice the usage of a few templates in the same site, with slight differences between them. The page-generation process can be seen as the result of the execution of some queries targeted at the support database, the source dataset being intertwined with HTML tags and other strings in a process that can be called codification. Eventually, URL links and banners or images can also be inserted.

In the past years various tools for data extraction have been proposed in the literature with the goal of efficiently solving this problem. Data published in the pages of very large sites and the higher rate of newcomer sites increased demand for semi-automatic or automatic systems. The cost of a reliable application whose maintenance requires human intervention and which will provide information with a high degree of precision increases linearly with the number of wrapped sources.

For example, RoadRunner ([Crescenzi et al., 2002a]) generates a schema for the data contained in many similar pages during an iterative process in which the current schema at step \( k \), \( T_k \) is tested and eventually updated against a new web page \( S \), resulting a new schema \( T_{(k+1)} \). The algorithm compares HTML tag structure of two or more pages from the same class (they are generated by the same script and are based on the same template). If the pages are not part of the same class the result will be a very general schema, which can extract data contained only in common structures, while the elements from a singular structure remain unknown because they cannot be identified. Based on the schema \( T \) a grammar is constructed, which is capable of recognizing among other things, nested-structured data objects with a variable number of values corresponding to their attributes.
2. Templates and Tree Edit Distance

Nowadays it is common for many web sites to contain large sets of pages, generated using a common template or layout. The values used to generate the pages (e.g., author, title, ...) are obtained from a database. The set of common characteristics from the structure of a web page is called template.

To our best knowledge there is no standard and formal definition of a template. [Bar-Yossef and Rajagopalan, 2002] provides the following template definition:

Definition 2.1 (Template) A template is a collection of pages that share the same look and feel and are controlled by a single authority.

A template will contain constants (elements that are not changing their value from a web page to another) and variables. Each variable of a template is an object which can contain important information. It is advisable that the automatically generated wrappers identify these objects.

In this paper, our goal is to analyze the web page structure in order to create clusters we have to find a metric (or distance) that will guide some traditional clustering algorithms. Since the web page structure is very well described by the DOM tree, a natural approach is to use the tree edit distance. This distance helps us compute the structural similarity value between two web pages. The string edit distance, the well-known example for the technique of dynamic programming, inspires the tree edit distance: intuitively the edit distance between two trees $T_A$ and $T_B$ is the cost of minimal set of operations needed for transforming $T_A$ into $T_B$.

For computing the edit distance we will have to consider three operations: (1) vertex removal, (2) vertex insertion and (3) vertex replacement.

Definition 2.2 (Vertex Operations) $Ren_T(l_{new})$ is a rename operation applied to the root of $T$ that yields the tree $T'$ with $\lambda(T') = l_{new}$ and first-level subtrees $T_1, T_2, ..., T_m$.

Given a node $x$ with degree 0, $Inst_T(x, i)$ is a node insertion operation applied to $T$ at $i$ that yields the tree $T'$ with $\lambda(T') = l$ and first-level subtrees $T_1, ..., T_i, x, T_{i+1}, ..., T_m$.

If a first-level subtree $T_i$ is a leaf node, $Del_T(T_i)$ is a delete node operation applied to $T$ at $i$ that yields the tree $T'$ with $\lambda(T') = l$ and first-level subtrees $T_1, ..., T_{i-1}, T_{i+1}, ..., T_m$.

We associate a cost to each of these operations; the sum of costs associated with edit operation defines the edit distance.

The solution to the problem consists in determining the minimal set of operations to transform one tree into another. An equivalent formulation of the problem is to discover the minimum cost mapping between the trees.

Definition 2.3 (Tree Mapping) Let $T_x$ be a tree and let $T_x[i]$ be the $i$-ism vertex of tree $T_x$ in a preorder walk of the tree.

A mapping between a tree $T_1$ of size $n_1$ and a tree $T_2$ of size $n_2$ is a set $M$ of ordered pairs $(i, j)$, satisfying the following conditions for all $(i_1, j_1), (i_2, j_2) \in M$:

1. $i_1 = i_2 \text{ iff } j_1 = j_2$
2. $T_1[i_1]$ is on the left of $T_1[i_2] \text{ iff } T_2[j_1]$ is on the left of $T_2[j_2]$
3. $T_1[i_1]$ is an ancestor of $T_1[i_2] \text{ iff } T_2[j_1]$ is an ancestor of $T_2[j_2]$.

Among many variations of algorithms for calculating the tree edit distance, we have limited to the top-down edit distance problem. After many experiments, for our implementation we have chosen the algorithm used for XML document categorization (as in [Nierman and Jagadish, 2002]) in order to cluster such documents. The tests effectuated with above mentioned algorithms lead us to this choice.

3. Process overview

The extraction process can be easily divided into four distinct steps:

1. Grouping the web pages into clusters ([Castro et al., 2004], [Crescenzi et al., 2002b], [Flesca et al., 2002]).
2. The generation of extraction pattern for each cluster. An extraction pattern is the generalization of all the pages contained in a cluster. See [Crescenzi et al., 2002a], [Wang and Lochovsky, 2002], [Liu et al., 2004] for more detailed descriptions.
3. Data alignment ([Wang and Lochovsky, 2002], [Zhai and Liu, 2005a]).
4. Data labeling (see [Wang and Lochovsky, 2002], [Arlotta et al., 2003]).

Our paper and software application deals with the first two actions.

3.1. Clustering algorithm

At the first step the labeled undirected complete graph $K_n = (V, U)$ is constructed, where:

1. $V = \{P_1, P_2, ..., P_n\}$ is the set of all HTML pages
The graph is complete. So for a set of cardinality \( n \) we will compute \( n \cdot (n - 1)/2 \) distance values.

At the second step, the algorithm computes the minimum spanning tree (MST) using one of the well-known algorithms: the Kruskal algorithm or the Prim algorithm. From this determined minimum spanning tree, we removed the edges whose associated cost \( w \) is greater than a value \( p \) \((w \geq p)\), where \( p \) is a superior limit apriori computed. Each conext component that remains after edges deletion represents a cluster.

### 3.2. Pattern generation

Next, we shall present some basic elements regarding the extraction pattern and the pattern generation. For more details the interested reader can consult [Castro et al., 2004]. An extraction pattern for trees is similar to a regular expression for strings. An extraction pattern is a rooted, ordered, labeled tree that can contain, as labels for vertices, special elements called wildcards. Each wildcard must be a leaf in the tree. There are four types of wildcards: single (\( \cdot \)) - a wildcard that captures one sub-tree and must be consumed, plus (+) - a wildcard that captures sibling sub-trees and must be consumed, option (\( ? \)) - a wildcard that captures one sub-tree and may be discarded, and kleene (*) - a wildcard that captures sibling sub-trees and may be discarded.

The goal is that every wildcard must correspond to a data-rich in the template: single and plus designate required objects, while option and Kleene wildcards designate optional objects. It is said that an extraction pattern accepts a tree if there is a mapping with a finite cost between the pattern tree and the data tree.

The process of pattern generation for a cluster of web pages consists of an iterating process called generalization, at each step the previous generated extraction pattern being composed with a DOM tree associated with a HTML page so as to result a new extraction pattern. At the end, the result is a pattern tree that accepts all the pages from the cluster. In the process of merging a pattern tree with a data tree, the tree edit distance algorithm is also used. There is defined a function with particular costs for insert, delete and replacement for all combination of wildcard operators with data nodes. For example, every wildcard followed by a set of option wildcards should be converted into a Kleene or plus wildcard that designates variable size objects.

More generally, it can be observed that optional vertices of the template that the extraction pattern is trying to harness should be kept optional after the composing step, and higher quantifiers should remain in the resulted pattern. The operation of replacing data nodes from the source tree with distinct data nodes from the target tree will be denoted with wildcard operations.

In conclusion, the pattern tree designates the minimum set of operations (insert, delete, replacement) for transforming one source tree into a target tree (or the minimum cost mapping between them).

### 4. Design and implementation

At this moment the application is composed of two independent modules: (1) a crawler and (2) a data analyzer.

The analyzer receives as input the structure of the site from the master index file, and a set of HTML pages. We transform each HTML page into a well-formed XHTML page. Of the applications that can be involved in this process, we enumerate JTidy, a Java tool based on [HTML Tidy] that is a W3C open source software, and [Neko HTML Parser].

From a constructed XHTML file, the DOM tree representation used in the next step of our process, can be created with no effort.

The input is split into a set of clusters: each cluster is composed of a set of HTML pages and its associated DOM trees, with the property that they share a similar template.

We have designed this module bearing in mind the idea of independency, in order to make it possible to be used in other applications.

Let’s denote by \( C = \{\text{cluster}_1, \text{cluster}_2, ..., \text{cluster}_n\} \) the set of resulted clusters. The clusters are represented as nodes in a graph \( G = (V, E) \). \( V = C \), and a directed edge from vertex \( i \) to vertex \( j \) exists iff there is at most one link from a page of cluster \( \text{cluster}_i \) to a page belonging to cluster \( \text{cluster}_j \).

In the next processing step, the analyzer extracts a pattern which will define the common template and will be used in the step of extracting data from web pages.

The Algorithm 1 describes the pattern matching process of data extraction from each HTML page. The ExtractData procedure determines a mapping between the pattern tree \( \text{pattern} \) and the data tree \( \text{input} \). Their root nodes represent these two trees. The current routine returns a list of mappings. If the two labels are identical (see line 2) then we shall try to find a matching between the sequence of children of node \( \text{pattern} \) and the sequence of children of node \( \text{input} \). The procedure ExtractDataFromList receives two lists of nodes as input: the algorithm is similar to that of aligning a string of wildcards with a string composed only of regular characters and returns the matchings. Then for each data node from children list of node \( \text{pattern} \) we try to find the first mapping that matches in labels (lines 5-14). GetChild(\( \text{input}, \text{label}, \text{index} \)) gets the child of the node \( \text{input} \) whose information is \( \text{label} \). The search for the child with this property starts from the position \( \text{index} \).
Algorithm 1 PatternMatch

Input: pattern - the root of the pattern tree (TreeNode)
       input - the root of the data tree

Output: the list of data extracted

Local: data - a temporary list
       k - the index of child node from where searching starts

1: procedure EXTRACTDATA(pattern, input)
2:     if pattern.getLabel() == input.getLabel() then
3:         data ← ExtractDataFromList(
4:             pattern.getChildren(), input.getChildren());
5:     k ← 1;
6:     for i ← 1, pattern.getChildrenCount() do
7:         label ← pattern.getChildAt(i).getLabel();
8:         if label is not wildcard then
9:             node ← GetChild(input, label, k);
10:            if node ≠ null then
11:                k ← input.getChildIndex(node) + 1;
12:                data.addAll(
13:                    ExtractData(pattern.childAt(i),
14:                        node));
15:         end if
16:     end for
17: else
18:     return data
19: end if
20: end procedure

5. Conclusions

We have developed a prototype tool for automatic data extraction from HTML pages. We shall pursue to improve the metric involved in the clustering step, and to study other different algorithms for computing tree edit distance. We aim at personalizing those algorithms for our special needs. The versatile design we have chosen easily allows changes of those elements.

References


