# Automatic Web News Extraction Using Tree Edit Distance

Davi de Castro Reis[1 2] Paulo B. Golgher[2] Altigran S. da Silva[3]
Alberto H. F. Laender[1]

[1]Computer Science Department
Federal University of Minas Gerais
Belo Horizonte, Brazil
{davi,laender}@dcc.ufmg.br

[2]Akwan Information Technologies
Av. Abraão Caram 430
Pampulha
Belo Horizonte, Brazil
{davi,golgher}@akwan.com.br

[3]Computer Science Department
Federal University of Amazonas
Manaus, Brazil
alti@dcc.fua.br

## ABSTRACT

The Web poses itself as the largest data repository ever available in the history of humankind. Major efforts have been made in order to provide efficient access to relevant information within this huge repository of data. Although several techniques have been developed to the problem of Web data extraction, their use is still not spread, mostly because of the need for high human intervention and the low quality of the extraction results. In this paper, we present a domain-oriented approach to Web data extraction and discuss its application to automatically extracting news from Web sites. Our approach is based on a highly efficient tree structure analysis that produces very effective results. We have tested our approach with several important Brazilian on-line news sites and achieved very precise results, correctly extracting 87.71% of the news in a set of 4088 pages distributed among 35 different sites.

## Categories and Subject Descriptors

H.3.m [**Information Storage and Retrieval**]: Miscellaneous—*Data Extraction, schema inference, Web*

## General Terms

Algorithms, Languages

## Keywords

data extraction, edit distance, trees, schema, electronic news

## 1. INTRODUCTION

Nowadays the Web poses itself as the largest data repository ever available in the history of humankind. Major efforts have been made in order to provide efficient access to relevant information within this huge repository. At least two broad views of this problem have evolved recently. The first one, characterized by the unstructured view of data, has developed breakthrough technologies (such as Web search engines) based on information retrieval [3] methods, which have been used in many successful commercial products. The second one, characterized by the structured or semi-structured view of data, borrows techniques from the database area to provide the means to effectively managing the data available on the Web [9]. Thus, several techniques have been adapted (or targeted specifically) to the problem of extracting data from the Web for further processing (querying, integration, mediation, etc.) [13]. However, these techniques are still not spread as the information retrieval based ones. This happens mostly because of two problems with these techniques: (1) the need for high human intervention and (2) the low quality of the extraction results. Thus, the motivation to develop new methods and tools to allow the effective deployment of a more structured view of the data available on the Web still remains.

Devising generic methods for extracting Web data is a complex (if not impossible) task, since the Web is very heterogeneous and there are no rigid guidelines on how to build HTML pages and how to declare the implicit structure of the Web pages. Thus, in order to develop effective methods for extracting Web data in a precise and completely automatic manner, it is usually required to take into account specific characteristics of the domain of interest. One of such domains is that of on-line newspapers and news portals on the Web, which have become one of the most important sources of up-to-date information. Indeed, there are thousands of sites that provide daily news in very distinct formats and there is a growing need for tools that will allow individuals to access and keep track of this information in a automatic manner.

In this paper, we present a domain-oriented approach to Web data extraction and discuss its application to automatically extracting news from Web sites. This approach is based on the concept of tree-edit distance [17, 20] and allows not only the extraction of relevant text passages from the pages of a given Web site, but also the fetching of the entire Web site content, the identification of the pages of interest (the pages that actually present the news) and the extraction of the relevant text passages discarding non-useful material such as banners, menus, and links.

To support this approach, we have developed a highly efficient tree structure analysis algorithm that outperforms, for practical purposes, the best results on tree-edit distance calculation in the literature. We have tested our approach with several important Brazilian on-line news sites and achieved very precise results, correctly extracting 87.71% of the news in a set of 4088 pages distributed among 35 different sites.

The rest of this paper is organized as follows. Section 2 gives an overview of the theory behind tree edit distance algorithms, the basis of our work. Section 3 presents our improved tree structure analysis algorithm, while Section 4 shows the application of this algorithm in the various tasks that comprise our approach. Experimental results demonstrating the effectiveness of our approach are in Section 5. Section 6 discusses related work. Finally, conclusions and directions for future work can be found in Section 7.


*WWW2004*, May 17–22, 2004, New York, New York, USA.
ACM 1-58113-844-X/04/0005.

## 2. TREE EDIT DISTANCE

The approach we have developed for finding and extracting data of interest from Web pages is based on the analysis of the structure of these target pages. More precisely, by evaluating the structural similarities between pages in a target site we are able to perform tasks such as grouping together pages with similar structure to form page clusters and finding a generic representation of the structure of the pages within a cluster. Indeed, as we shall see, such tasks are key to our approach.

Since the structure of a Web page can be nicely described by a tree (e.g., a DOM tree), we have resorted to the concept of *tree edit distance* [17, 20] to evaluate the structural similarities between pages. Intuitively, the edit distance between two trees $T_A$ and $T_B$ is the cost associated with the minimal set of operations needed to transform $T_A$ into $T_B$. In this section we review this important concept along with its related formalisms and describe how we use it to analyze the structure of Web pages.

Trees are one of the most common data structures used in computer science. Formally, they are defined as directed acyclic simple graphs. Although most of the discussion in this section can be generalized to deal with different types of tree, we are interested only in one specific type of tree, called *labeled ordered rooted tree*. A rooted tree is a tree whose root vertex is fixed. Ordered rooted trees are rooted trees in which the relative order of the children is fixed for each vertex. Labeled ordered rooted trees have a label $l$ attached to each of their vertices. Figure 1 shows an example of such a tree. From now on, we refer to labeled ordered rooted trees simply by trees, except when explicitly stated.

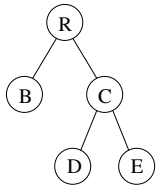


**Figure 1: A labeled ordered rooted tree with root $R$**

In its traditional formulation, the tree edit distance problem considers three operations: (a) vertex removal, (b) vertex insertion, and (c) vertex replacement. To each of of these operations, a cost is assigned. The solution of this problem consists in determining the minimal set of operations (i.e., the one with the minimum cost) to transform one tree into another. Another equivalent (and possibly more intuitive) formulation of this problem is to discover a *mapping* with minimum cost between the two trees. The concept of mapping (introduced in [18]) is formally defined next.

DEFINITION 1. *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 $(i_1, j_1), (i_2, j_2) \in M$*

- *$i_1 = i_2$ iff $j_1 = j_2$;*
- *$T_1[i_1]$ is on the left of $T_1[i_2]$ iff $T_2[j_1]$ is on the left of $T_2[j_2]$;*
- *$T_1[i_1]$ is an ancestor of $T_1[i_2]$ iff $T_2[j_1]$ is an ancestor of $T_2[j_2]$.*

In Definition 1, the first condition establishes that each vertex can appear no more than once in a mapping, the second enforces order preservation between sibling nodes and the third enforces the hierarchical relation between the nodes in the trees. Figure 2 illustrates a mapping between two trees.

Intuitively, a mapping is a description of how a sequence of edit operations transform a tree into another, ignoring the order in which these operations are applied. In Figure 2, a dotted line from a vertex of $T_1$ to vertex of $T_2$ indicates that the vertex of $T_1$ should be changed if the vertices are different, remaining unchanged otherwise. Vertices of $T_1$ not touched by dotted lines should be deleted, and vertices of $T_2$ not touched should be inserted.

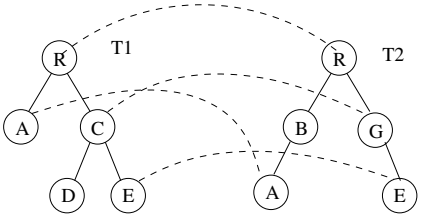


**Figure 2: A mapping example**

As we have already mentioned, estimating a tree edit distance is equivalent to finding the minimum cost mapping. Let $M$ be a mapping between tree $T_1$ and tree $T_2$, let $S$ be a subset of pairs $(i, j) \in M$ with distinct labels, let $D$ be the set of nodes in $T_1$ that do not occur in any $(i, j) \in M$ and let $I$ be the set of nodes in $T_2$ that do not occur in any $(i, j) \in M$. The mapping cost is given by $c = Sp + Iq + Dr$, where $p$, $q$ and $r$ are the costs assigned to the replacement, insertion, and removal operations, respectively. It is common to associate a unit cost to all operations, however, specific applications may require the assignment of distinct costs to each type of operation.[1]

The tree edit distance problem is a difficult one, and several algorithms, with different tradeoffs, have been recently proposed, but all formulations have complexities above quadratic [6]. Further, it has been proved that, if the trees are not ordered, the problem is NP-complete [27]. The first algorithm for the mapping problem was presented in [18], and its complexity is $O(n_1 n_2 h_1 h_2)$, where $n_1$ and $n_2$ are the sizes of the trees and $h_1$ and $h_2$ are their heights. This is a dynamic programming algorithm that recursively calculates the edit distance between the strings formed by the sets of children vertices of each internal vertex in the tree. In [21], a new algorithm was presented with cost $O(d^2 n_1 n_2 min(h_1, l_1) min(h_2, l_2))$, where $d$ is the edit distance between the trees and $l_1$ and $l_2$ are the number of leaves in each tree. Notice that this cost depends on the algorithm output. The best known upper limit for this problem is due to an algorithm presented in [6] with complexity $O(n_1 n_2 + l_1^2 + l_1^{2.5} l_2)$.

Despite the inherent complexity of the mapping problem in its generic formulation, there are several practical applications that can be modelled using restricted formulations of it. By imposing conditions to the basic operations corresponding to the original formulation in Definition 1 (i.e., replacement, insertion and removal), four classical restricted formulations are obtained: *alignment*, *distance between isolated trees*, *top-down distance*, and *bottom-up distance*, for which more convenient and fast algorithms have been proposed [19, 22].

Detailing each one of these formulations and algorithms is beyond the scope of this paper, but since our approach is based on a restricted version of the top-down mapping problem, we will briefly review and illustrate it. Informally, a top-down mapping restricts the removal and insertion operations to take place only in the leaves of the trees. Figure 3 illustrates a top-down mapping which is formally defined as follows.

[1]Other applications may even require a distinct set of operations.

DEFINITION 2. *A mapping $M$ between a tree $T_1$ and a tree $T_2$ is said to be* top-down *only if for every pair $(i_1, i_2) \in M$ there is also a pair (parent($i_1$),parent($i_2$)) $\in M$, where $i_1$ and $i_2$ are non-root nodes of $T_1$ and $T_2$ respectively.*

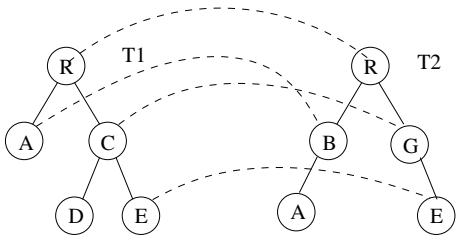


**Figure 3: A top-down mapping example**

The first algorithm for the top-down edit distance problem was proposed by Selkow [17]. In [25], Yang presents a recursive dynamic programming algorithm with cost $O(n_1 n_2)$ for the problem, where $n_1$ and $n_2$ are the sizes of $T_1$ and $T_2$, respectively.

One of the most popular algorithms for the problem is presented in [5] also with cost $O(n_1 n_2)$. This algorithm, however, is not recursive and the problem is solved within a single dynamic programming instance. The paper also presents an external memory variation for this algorithm.

Top-down mappings have been successfully applied to several Web related applications such as document categorization. For instance, Nierman and Jagadish [16] use a top-down distance algorithm to cluster XML documents.

In our case, we are interested in the problem of evaluating the similarity between Web pages. Indeed, most Web pages are structured according to formats such as HTML and XML which, as mentioned before, can be seen as labeled ordered rooted trees [7]. Actually, the DOM paradigm, commonly used for manipulating Web pages, uses this tree representation.

In the next section, we present a new algorithm for determining a restricted form of top-down mapping between two trees that represent Web pages and, as a consequence, the tree edit distance between them.

# 3. THE RTDM ALGORITHM

In this section, we present an algorithm for determining a new type of mapping that we call *Restricted Top-Down Mapping*. Intuitively, in the restricted top-down mapping, besides the insertion and removal operations, the replacement operation of different vertices is also restricted to the leaves of the trees. More formally, we have the following definition.

DEFINITION 3. *A top-down mapping $M$ between a tree $T_1$ and a tree $T_2$ is said to be* restricted top-down *only if for every pair $(i_1, i_2) \in M$, such that $t_1[i_1] \neq t_2[i_2]$, there is no descendent of $i_1$ or $i_2$ in $M$, where $i_1$ and $i_2$ are non-root nodes of $T_1$ and $T_2$ respectively.*

Figure 4 shows a restricted top-down mapping. As done for the family of edit distances mentioned before, we can define the *restricted top-down edit distance* between two trees $T_1$ and $T_2$ as the cost of the restricted top-down mapping between the two trees.

The *RTDM* algorithm combines the ideas presented in [25] and [19]. To determine the restricted top-down mapping between two trees $T_1$ and $T_2$, the *RTDM* algorithm first finds all identical subtrees that occur at the same level of the input trees. This step is performed in linear time using a graph of equivalence classes, in a similar way to what is done in [19]. Our algorithm, however, is based on a post-order traversal of the trees. We can use this much simpler approach because we only look for the identical sub-trees of the *same level*. This first step of the algorithm has linear cost, with respect to the number of vertices in the trees.

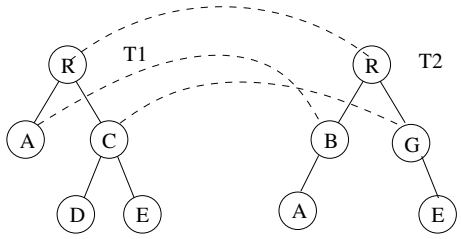


**Figure 4: A restricted top-down mapping example**

Once the vertices in the trees are grouped in equivalent classes, an adaptation of Yang's algorithm [25] is applied to obtain the minimal restricted top-down mapping between the trees. This second step of the algorithm is shown in Figure 5. This figure only shows the algorithm version for calculating the tree edit distance, but its modification for obtaining the mapping is straightforward.

As we have already mentioned, the traditional top-down edit distance algorithm by Chawathe [5] has a complexity of $O(n_1 n_2)$ for all cases, that is, the best, the expected and the worst cases. The *RTDM* algorithm also has a worst case complexity of $O(n_1 n_2)$, but, in practice, it performs much better due to the fact that it only deals with restricted top-down mappings.

The worst case of the *RTDM* algorithm occurs when the two trees being compared are all identical, except for their leaves. In all other cases, the cost is amortized by the short-cuts in lines $20-25$, which we call the *top-down short-cut*, or in lines $17-18$, which we call the *bottom-up short-cut*. Further, when all we want to know is whether the tree edit distance is under a given threshold, the short-cut in lines $15-16$ prevents the recursion to continue. This is a very common situation when we need to cluster trees based on their structural similarities.

We also notice that we can trivially alter lines $20-25$ (the so-called *top-down short-cut*), to create an algorithm that determines the traditional (i.e., non-restricted) top-down edit distance. Thus, we also have a new algorithm for the traditional formulation of the problem.

Another interesting aspect of the *RTDM* algorithm is its flexibility with respect to the cost of the edit operations. This property allows using the algorithm in more complex derivations of the problem. For instance, it allows comparing a given tree instance with a tree pattern of variable size. This problem is analogous to the problem of matching regular expressions with strings and has been addressed in the literature [26].

In the next section, we show how the *RTDM* algorithm can be applied to the problem of automatically finding news available on Web sites and extracting their components (e.g., titles, body, etc.) for further processing.

# 4. AUTOMATICALLY EXTRACTING WEB NEWS

In this section we discuss a Web news extraction approach that relies on the *RTDM* algorithm to identify relevant text passages containing news and their components, extract them and discard useless material such as banners, links, etc. Our approach has basically two main tasks: (1) the crawling of news portals to fetch the pages of interest and (2) the extraction of the news from the

```
1  RTDM(T1, T2, ε: threshold)
2  begin
3    let m be the number of children of T1 root
4    let n be the number of children of T2 root
7    M[i, 0] ← 0 for all i = 0, ..., m
8    M[0, j] ← 0 for all j = 0, ..., n
9    for i = 1 to m
10     for j = 1 to n
11       C_i ← descendents(t1[i])
12       C_j ← descendents(t2[j])
13       d ← M[i − 1, j] + Σ_k^{t1[k]∈C_i} delete(t1[k])
14       i ← M[i, j − 1] + Σ_k^{t2[k]∈C_j} insert(t2[k])
15       if M[i − 1, j − 1] > ε
16         s ← ∞
17       elsif t1[i] and t2[j] are identical sub-trees
18           s ← 0
19       elsif
20           if t1[i] is a leaf
21             s ← replace(t1[i], t2[j])
22             s ← s + Σ_k^{t2[k]∈C_j} insert(t2[k])
23           elsif t2[j] is a leaf
24               s ← replace(t1[i], t2[j])
25               s ← s + Σ_k^{t1[k]∈C_i} delete(t1[k])
26           else
27               s ← RTDM(t1[i], t2[j], ε)
28           fi
29       fi
30       M[i, j] ← min(d, i, s);
31     end
32   end
33   return M[m, n]
34   end
```

**Figure 5: The *RTDM* Algorithm. The functions *replace*, *delete* and *insert* give the costs of vertex replacement, vertex removal and vertex insertion, respectively**

HTML pages collected. Since Web crawling techniques have been extensively discussed elsewhere [12], we focus our discussion on the extraction task, in which resides most of our contributions.

To extract the desired news, our approach recognizes and explores common characteristics that are usually present in news portals. For instance, most news sites have the following organization: (a) a home page that presents some headlines, (b) several section pages (or channels) that provide the headlines divided in areas of interest (e.g., sports, technology, international, etc.), (c) pages that actually present the news, containing the title, author, date and body of the news. The goal of our approach is to correctly extract the news, disregarding the other pages.

Our approach relies on the basic assumption that the news site content can be divided in groups that share common format and layout characteristics. This is rather a safe assumption, since nowadays most of the Web content is built using programs or scripts that read the content from a database, format it, and generate the output as an HTML page. We call this set of common layout and format features a *template*. Figure 6 presents two different templates available in the CNN site.

DEFINITION 4. *A* template *is the set of common layout and format features that appear in a set of HTML pages that is produced by a single program or script that dynamically generates the HTML page content.*

In the case of news, templates are filled by journalists, usually through the use of specific Web applications or some database interface. Each field of a template (e.g., a news title) we call a *data-rich object*. Ideally, the extractors generated by our approach should be able to identify each one of these data-rich objects, and discover, among them, which ones correspond to the title and the body of the news.

According to our approach, the extraction task is performed in four distinct steps: (1) page clustering, (2) extraction pattern generation, (3) data matching and (4) data labeling. Figure 7 illustrates these steps.

In the following sections, we detail each step that comprises the extraction task. We notice that our approach is simple and orthogonal, once the core of the main steps (clustering, extraction and matching) is the *RTDM* algorithm, with variations on the cost model for the edit operations.

## 4.1 Page Clustering

This first step takes as input a previously crawled set of pages (a training set) and generates clusters of pages that share common formating/layout features, i.e., share the same template. Each cluster is later generalized into an extraction structure for a template, in the extraction pattern generation step. Notice that the cluster algorithm cannot simply group pages by their address (URL), because subtle changes in script or cgi parameters may result in a completely different HTML page.

To generate the clusters, we use traditional hierarchical clustering techniques [23] in which the distance measured is the output of our *RTDM* algorithm. There are no pre-defined number of clusters. Instead, we adopt a constant threshold to determine if two given clusters should be merged. In our implementation we used 80% of similarity as the threshold value. The cost model for this step is the simplest one. Every vertex insertion, removal or replacement has unit cost. The replacement of equally labeled vertices has cost zero. Other works [16] suggest a more sophisticated set of operations, but our experiments have shown that this simple model is effective for our purposes. The output of this step is a set of page clusters that share the same template.

## 4.2 Extraction Pattern Generation

In this step, our approach generalizes a cluster of pages into what we call a *node extraction pattern (ne-pattern)*. Formally, an ne-pattern is a tree defined as follows.

DEFINITION 5. *Let a pair of sibling sub-trees be a pair of sub-trees rooted at sibling vertices. A* node extraction pattern *is a rooted ordered labeled tree that can contain special vertices called wildcards. Every wildcard must be a leaf in the tree, and each wildcard can be of one of the following types:*

- **SINGLE** *(·) 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.*
- **KLEENE** (∗) *A wildcard that captures sibling sub-trees and may be discarded.*

We can think of an ne-pattern as a kind of regular expression for trees. We call a wildcard every vertex in the tree that can match any symbol (any label) with its associated type. Our purpose in this step is to assure that each wildcard corresponds to a data-rich object in the template. Single and plus wildcards should correspond to


**Figure 6: Some templates available in the CNN site**

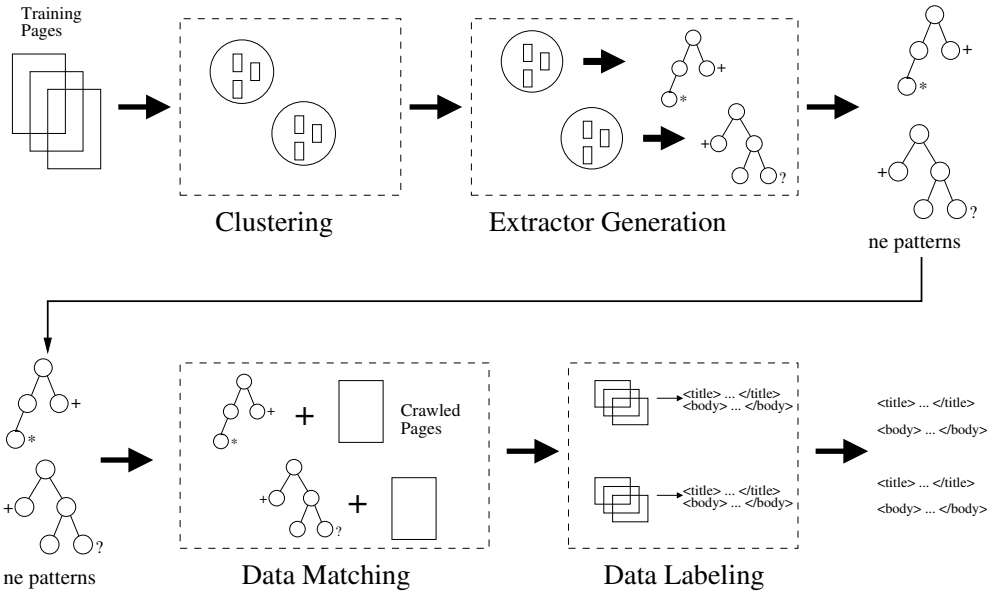


**Figure 7: The main extraction steps**

required objects, such as the title of a news, and option and Kleene wildcards should correspond to optional objects, such as related news lists.

Further, we say that an ne-pattern *accepts* (or matches) a given tree if there is a mapping with no infinite cost between the ne-pattern and the target tree. We define formally this concept and the cost model associated with this mapping in Section 4.3.

The goal of this step in the extraction task is, taking as input a page cluster, to generate an ne-pattern that accepts all the pages in this cluster. Thus, the content differences between the pages in the cluster are modeled as wildcards in our ne-patterns. To generate such ne-patterns, we rely on what we call a *composition operation*, defined as follows.

DEFINITION 6. *Let $T_1^x$ and $T_2^x$ be distinct ne-patterns. Then the composition of $T_1^x$ and $T_2^x$, $T_1^x \circ T_2^x$, is a ne-pattern $T_3^x$ such that:*

- *Let $S_1$ be the set of trees accepted by $T_1^x$.*
- *Let $S_2$ be the set of trees accepted by $T_2^x$.*
- *Let $S_3$ be the set of trees accepted by $T_3^x$.*
- *Then $S_1 \cup S_2 \subseteq S_3$.*

The process of generating an ne-pattern consists of iterating all the trees that represent the pages in the cluster and incrementally composing one to each other in the cluster. Notice that any tree can be seen as an ne-pattern without any wildcard. At the end of the process, we have an ne-pattern that accepts all pages in that cluster.

Let us see how we can use the *RTDM* algorithm to implement the composition operation. First, we say that vertices $a$ and $b$ of an ne-pattern are equal if and only if:

- $a$ and $b$ are wildcards and both are of the same type;
- $a$ and $b$ are not wildcards and the labels associated with $a$ and $b$ are equal.

This is the equality operator for the *RTDM* algorithm. As a cost model, we give the same weight, 1, to any edit operation in the

trees. Given two ne-patterns $T_1^x$ and $T_2^x$, we use the *RTDM* algorithm to obtain a mapping $M_{T_1^x \to T_2^x}$. From this mapping, we create the composite ne-pattern $T_3^x = T_1^x \circ T_2^x$ using the following rules:

- if $a$ is not in the mapping, then add $a'$ to $T_3^x$ where $a' = f(a, ?)$;
- if $a$ maps to $b$ then add $a'$ to $T_3^x$ where $a' = f(a, b)$;
- and $f(a, b)$ is defined as:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $f(*,*)$ | $=$ | $*$ | $f(+,+)$ | $=$ | $+$ | $f(.,.)$ | $=$ | $.$ |
| $f(*,+)$ | $=$ | $*$ | $f(+,.)$ | $=$ | $+$ | $f(.,?)$ | $=$ | $?$ |
| $f(*,?)$ | $=$ | $*$ | $f(+,?)$ | $=$ | $*$ | $f(.,n)$ | $=$ | $.$ |
| $f(*,.)$ | $=$ | $*$ | $f(+,n)$ | $=$ | $+$ | $f(?,?)$ | $=$ | $?$ |
| $f(*,n)$ | $=$ | $*$ | | | | $f(?,n)$ | $=$ | $?$ |

$f(n_1, n_2) = n_1$ if $n_1$ and $n_2$ have identical labels
$f(n_1, n_2) = .$ if $n_1$ and $n_2$ have different labels

where $n$, $n_1$, $n_2$ are non-wildcard vertices and the parameter order is not relevant.

The motivation behind this set of operations is that optional vertices of the template that the ne-pattern is trying to model should be kept optional after composing the ne-pattern with a new tree, and higher quantifiers (i.e., Kleene and plus) should be kept in the final ne-pattern. Non-wildcard vertices in the ne-pattern that are mapped to different (as defined by our equality operator) non-wildcard vertices in the tree being composed should result in new wildcards.

We notice that some data-rich objects in the pages might span through several sibling sub-trees, like a text of a news body that is composed of many adjacent paragraphs. Capturing each of these objects as a single entity is the purpose of the plus and Kleene wildcards.

If we look carefully at the definition of the function $f(a, b)$ above, we will see that there is no wildcard quantifier "promotion" policy, or, in other words, wildcards plus and Kleene will never be generated if there are no plus or Kleene wildcards in the input of the function. These wildcards are created in a post-processing step whenever we compose two ne-patterns.

This post-processing is actually quite simple. Every wildcard followed by a set of option wildcards should be converted into a wildcard for variable size objects, that is, Kleene or plus wildcards. If the wildcard before the set of option wildcards is a single or a plus wildcard, then the set of option wildcards and the precedent wildcard are converted to a plus wildcard. If the wildcard is an option or Kleene wildcard, then both this wildcard and the adjacent option wildcards are converted to a Kleene wildcard. Figure 8 illustrates the whole ne-pattern generation task, including the "promotion" of a wildcard. In our approach, even if wildcards are separated by a maximum of 3 non-wildcard vertices they can be merged (including the non-wildcard vertices) into a single variable size wildcard (plus or Kleene).

## 4.3 Data Matching

In this step, our approach matches the set of generated ne-patterns to the set of recently crawled pages. To find the most appropriate ne-pattern to a crawled HTML page, we again rely on our *RTDM* algorithm.

Before discussing the cost model for the matching step, we need to understand what the intuition behind the matching of the ne-patterns is. In this context, we say that, in a given mapping, if one wildcard vertex in the ne-pattern maps to a vertex in the target HTML tree, then the wildcard *consumes* the vertex. Now let us define the desired behavior for a mapping between the ne-pattern and the target tree, so that we can create an appropriate cost model.

DEFINITION 7. *We define a* match *between an ne-pattern and a target HTML tree as a mapping such that the following rules are satisfied in this order:*

1. *Every non-wildcard vertex in the ne-pattern must map to an identical vertex in the target tree.*
2. *Every vertex in the target tree must map to an identical non-wildcard vertex in the ne-pattern or be consumed by a wildcard.*
3. *Single wildcards (.) must consume one sub-tree of the target tree.*
4. *Plus wildcards (+) must consume at least one sub-tree of the target tree.*
5. *Option wildcards (?) must consume one sub-tree of the target tree, if it is possible.*
6. *Kleene wildcards (∗) must consume at least one sub-tree of the target tree, if it is possible.*
7. *Plus wildcards (+) must consume as many sibling sub-trees of the target tree as possible.*
8. *Kleene wildcards (∗) must consume as many sibling sub-trees of the target tree as possible.*

The satisfaction of Rules 1, 2, 3 and 4 is enough to guarantee that the ne-pattern *accepts* the target tree. Rules 5 and 6 assure that the match is as *tight* as possible, or, if it is possible to use an optional wildcard without violating the acceptance condition, it must be used. Rules 7 and 8 are always automatically satisfied, and are declared to help understanding the behavior of the ne-pattern.

The equality function for the *RTDM* algorithm is very simple. Non-wildcard vertices with identical labels are equal and the equality comparison with a wildcard vertex always fails. Let $a$ be a vertex in the ne-pattern, and $b$ a vertex in the target tree. We define the cost model for the *RTDM* algorithm as follows:

- **Vertex Replacement**
  - (A) $a$ is a wildcard $\to 0$
  - (B) else $\to \infty$
- **Vertex Insertion**
  - (C) There is an ancestor of $b$ such that it is consumed by a wildcard $\to 0$
  - (D) The left sibling of $b$ is consumed by a $* \to 0$
  - (E) The left sibling of $b$ is consumed by a $+ \to 0$
  - (F) else $\to \infty$
- **Vertex Removal**
  - (G) $a =?$ or $a = * \to 1$
  - (H) else $\to \infty$

The replacement cost (A) guarantees that only wildcards can be replaced by the sub-trees they consume. The insertion cost (C) allows complete sub-trees to be consumed by the wildcards. Costs (D) and (E) allow wildcards to consume lists of sibling sub-trees. The vertex removal cost (G) assures that only optional wildcards can be deleted, and it associates a non-zero cost with the deletion of an optional wildcard, so they are preferably covered by cost (A). Finally, costs (B), (F) and (H) together guarantee that the ne-pattern must *accept* the target page, or the mapping will have infinite cost.

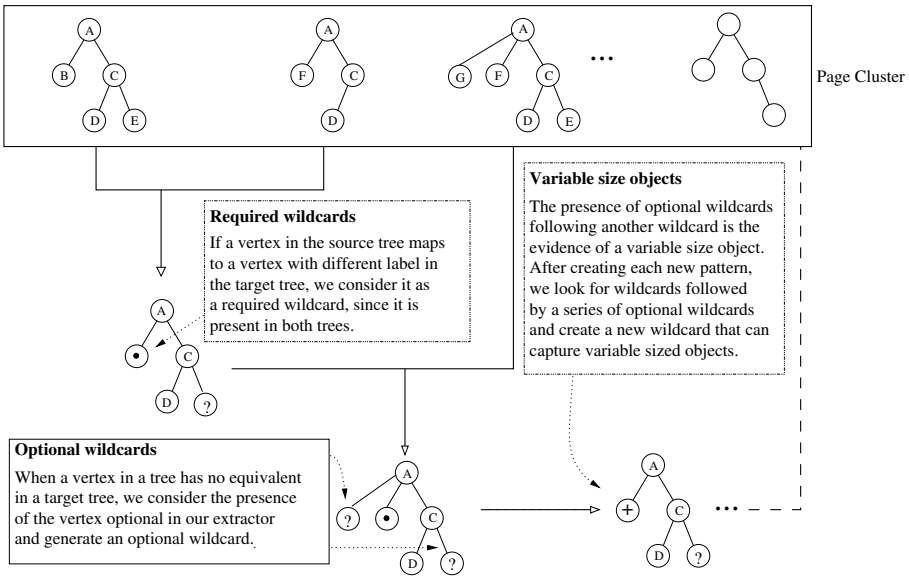


**Figure 8: How an *ne-pattern* is created from a cluster of similar pages**

Although costs (C), (D) and (E) seem quite complicated at a first glance, they are trivially implemented in constant time. To check the validity of any of them, we just need to check if the vertex in the target tree is being inserted in the position of (or immediately after) a wildcard in the ne-pattern. This cost model guarantees that either the conditions in Definition 7 are satisfied or the mapping has infinite cost.

Once the ne-pattern has been selected, the extraction process is straightforward. Both trees (the ne-pattern and the HTML page) are traversed in pre-order and for each wildcard found in the ne-pattern, the text passage in the vertices consumed by the wildcard is extracted from the HTML page. Figure 9 illustrates the matching process.

## 4.4 Data Labeling

The output of the data matching step is a set of ordered text passages, each one corresponding to a set of vertices consumed by a ne-pattern wildcard. More formally, we can define the output of a match as a set $T = (t_1, p_1), (t_2, p_2), ..., (t_n, p_n)$ where each $t_i$ is a text passage retrieved by a wildcard and $p_i$ is the vertex position of this wildcard if we perform a pre-order traversal of the ne-pattern.

The goal of the data labeling step is to select from $T$ the passages $t_i$ and $t_j$ that correspond to the title and the body of the news being extracted from the Web page[2]. To achieve this, we apply simple heuristics to $T$ as discussed bellow. Given a set of extracted passages $T = (t_1, p_1), (t_2, p_2), ..., (t_n, p_n)$ we say that:

- $length(t_i)$ is the number of terms (words) in passage $t_i$;
- $\mid t_k \cap t_i \mid$ is the number of terms that occur in passages $t_k$ and $t_i$;
- $t_i$ is a news body iff $length(t_i) > length(t_k)\ \ \forall\ 1 < k < n, k \neq i$ and $length(t_k) > 100$ (Body labeling heuristics);
- $t_j$ is a news title iff $1 \leq length(t_j) \leq 20$ and $\frac{|t_j \cap t_i|}{p_j - p_i} > \frac{|t_k \cap t_i|}{p_k - p_i}\ \forall\ 1 < k < i, k \neq j$ (Title labeling heuristics).

[2]In this paper we do not focus on the extraction of the news date, because we can trivially determine it from the date the news first appeared on the Web site.

In other words, the passage elected to be the body of the news is the longest one with more than 100 words. Further, the passage selected to be the title is one that has ranges from 1 to 20 words, has a maximum intersection with a body passage, and is the closest one to the body. The intuition behind the title selection is that most of the times the title is placed near the body and its terms usually appear in the news body.

Despite using this simple heuristics, our labeling strategy is very effective, as shown next by our experiments.

# 5. EXPERIMENTAL RESULTS

## 5.1 Setup

Our experiments were run using 4088 HTML pages collected from 35 different Brazilian on-line news sites. The sites chosen are the most popular vehicles from the Brazilian press, including country-wide newspapers, news agencies, magazines and main regional publications. All the experiments were carried out using a 700MHz Pentium III processor with 128MB of RAM.

## 5.2 The RTDM algorithm

Considering that the *RTDM* algorithm is the basis of the news extraction approach described in this paper, we must assure that it runs fast and scales well. To the best of our knowledge, there is no other restricted top-down mapping algorithm in the literature, so we decided to compare the *RTDM* algorithm with the competitive top-down edit distance algorithm presented by Chawathe in [5].

Adapting Chawathe's algorithm to the extraction pattern generation and data matching steps of our approach is not trivial, but the page clustering step can be easily adapted to use this algorithm. Thus, we built two versions of the clustering step, one powered by the *RTDM* algorithm, and the other one powered by Chawathe's. Comparing the executions times, the *RTDM* algorithm in general outperforms the alternative algorithm by 4 times, but sometimes it is more than 10 times faster. The main disadvantage of Chawathe's algorithm is that it is always quadratic with respect to the number of vertices of the trees being compared. Figure 10 shows how the algorithms perform when the average number of vertices of the trees being compared increases. If we analyze the behavior of the

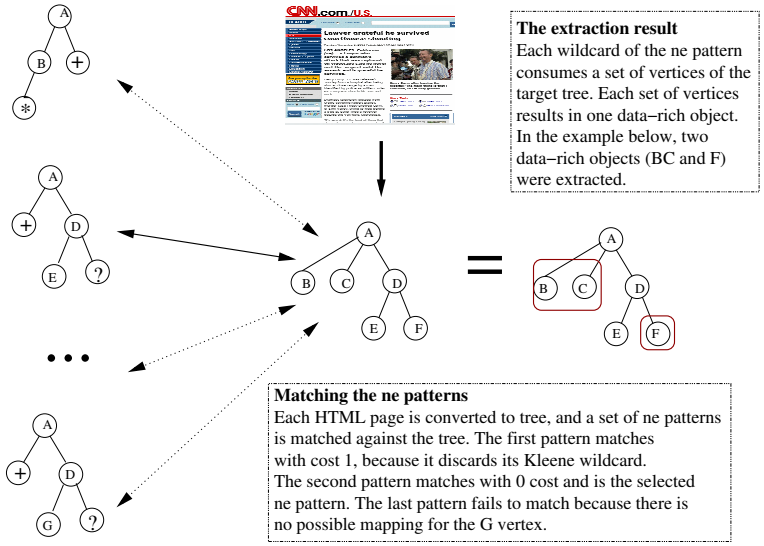


**Figure 9: How ne-patterns are matched with Web pages**

*RTDM* algorithm when the number of vertices grows, we see that it depends not only on the number of vertices in the trees, but also on the properties of the trees. This is due to the several short-cuts that the algorithm uses to avoid recursively checking the complete trees and to the different properties of the restricted top-down mappings. Each point in Figure 10 roughly corresponds to a cluster.

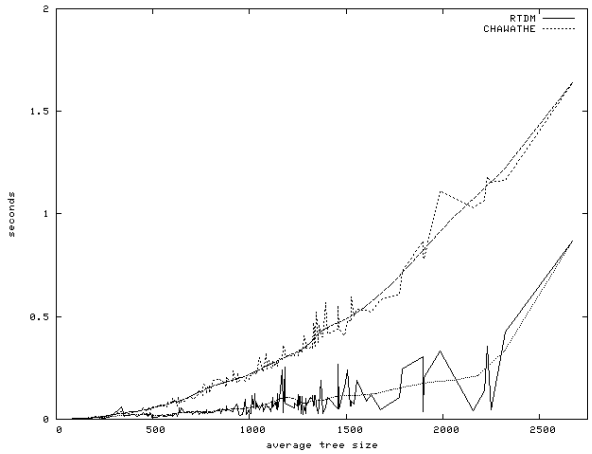


**Figure 10: RTDM and Chawathe's algorithm - the bezier approximation of the curves shows that Chawathe's algorithm has quadratic growth**

## 5.3 News Extraction

The second part of our experiments consisted of analyzing the output of the complete extraction process. We manually compared the extracted news with the original HTML pages, to check for their correction and completeness. Table 1 presents the results for all 35 sites. Our approach was able to extract correctly an average of 87.71% of the news, while 9.25% were erroneously extracted and 3.04% were not extracted.

During our experiments, we noticed that the use of restricted top-down mappings is really suitable for identifying data-rich portions of Web pages. In the data labeling step, however, it is still difficult to precisely identify the title of the news. Most of the errors were due to subtitles and authors names that were misidentified as titles. Despite this, we achieved very good results with a completely automatic approach and simple labeling heuristics.

Even though we have used simple labeling heuristics, the reason for this high level of effectiveness is that, after the extraction of the data-rich portions of the pages, the size of the set of candidates text passages for title and body is usually reduced from a range of hundreds to thousands to a range of two to five candidates.

# 6. RELATED WORK

One of the reasons why the Web has achieved its current huge volume of data is the fact that a great and increasing number of data-rich Web sites have their pages automatically generated from databases. Taking advantage of this, a number of approaches have been recently proposed to analyze the structure of the pages of these Web sites with the purpose of inferring a general data schema for them and ultimately generating wrappers to extract this data.

The first solution for this problem was proposed by Grumbach and Mecca [11] assuming the existence of collections of data-rich pages bearing a similar structure or schema. In [7], an algorithm is proposed to infer union-free regular expressions that represent page schemas. For complex schemas with optional attributes, the algorithm execution can explode and thus it is considered as having exponential cost [7]. By using several heuristics, Arasu and Garcia-Molina [1] have recently proposed a polynomial time algorithm for the problem. Since the approaches proposed in [1] and [7] require no human intervention, an important problem that they have left open is how to automatically label the extracted data. This problem is addressed in [2] but the solution proposed is not general enough.

There are also several works in the literature that address the problem of schema extraction from collections of XML documents. The XTRACT system [10] uses MDL, an information theory technique, to infer concise and accurate schemas from a collection of XML documents. Min et al. presented a much faster system, with better results in [15]. Although we do not directly consider the schema extraction problem in this work, the ne-patterns we generate resemble schema definitions, and we believe that the techniques proposed here can also be applied to the schema extraction problem. Also, the ideas behind the XML schema extraction systems can be used to improve our work in situations in which the data

| Site | √ | × | Not Extracted | # pages |
|---|---|---|---|---|
| A notícia Joenville | 83.95% | 13.58% | 2.47% | 81 |
| AOL Brasil | 87.60% | 12.40% | 0.00% | 121 |
| Agência Estado | 94.90% | 4.08% | 1.02% | 98 |
| Correio Brazilense | 71.43% | 11.90% | 16.67% | 119 |
| Correio da Bahia | 98.15% | 1.85% | 0.00% | 54 |
| DCI | 96.55% | 0.00% | 1.72% | 228 |
| Diário de Natal | 96.62% | 0.00% | 2.90% | 206 |
| Diário Grande ABC | 100.00% | 0.00% | 0.00% | 8 |
| Diário do Maranhão | 75.00% | 25.00% | 0% | 48 |
| Diário Popular | 100.00% | 0.00% | 0.00% | 85 |
| Diário de Cuiaba | 85.26% | 12.82% | 1.92% | 154 |
| Diário do Com. BH | 92.31% | 3.85% | 3.85% | 26 |
| Estado de Minas | 77.40% | 21.47% | 1.13% | 177 |
| Estado de São Paulo | 84.33% | 15.21% | 0.46% | 217 |
| Folha de Pernam. | 91.18% | 1.47% | 7.35% | 68 |
| Folha de São Paulo | 77.78% | 13.33% | 8.89% | 225 |
| Gazeta Digital | 88.17% | 10.75% | 1.08% | 185 |
| Gazeta Mercantil | 87.01% | 0.65% | 12.34% | 154 |
| Hoje em Dia | 90.91% | 9.09% | 0.00% | 66 |
| IDG Now | 93.18% | 2.27% | 4.55% | 44 |
| ITWeb | 96.88% | 0.00% | 3.13% | 32 |
| InvestNews | 95.47% | 0.00% | 4.53% | 329 |
| Jornal da Tarde SP | 90.57% | 5.66% | 3.77% | 159 |
| O Dia RJ | 75.86% | 22.07% | 2.07% | 144 |
| O Globo | 99.35% | 0.65% | 0% | 307 |
| Tribuna Santos | 75.00% | 22.58% | 2.42% | 123 |
| Tribuna da Bahia | 81.13% | 15.09% | 3.77% | 53 |
| Tribuna da Imprensa | 90.63% | 9.38% | 0% | 32 |
| UOL | 74.53% | 23.58% | 1.89% | 106 |
| Valor On Line | 91.45% | 4.27% | 4.27% | 117 |
| Verdade On Line | 82.61% | 13.04% | 4.35% | 22 |
| Vox News | 80.00% | 0.00% | 20.00% | 35 |
| Yahoo | 93.64% | 0.91% | 5.45% | 208 |
| Zero Hora | 83.22% | 16.11% | 0.67% | 149 |
| Total | 87.71% | 9.25% | 3.04% | 4088 |

**Table 1: Results obtained for the news extraction process.**

being extracted is ruled by more complex schemas than those of found in on-line news. Actually, the problem of schema extraction for Web pages has been proven NP-Complete recently [24].

The automatic classification of Web pages based on their structure is addressed in [8]. However, this work differs from ours since in our case the classification is based on the structural properties of the pages and not on the results of the wrapping process.

The ChangeDetector$^{TM}$ system [4] uses an algorithm very similar to ours in its entity-based change detection step. The algorithm, however, works with hashes of the contents of the subtrees, falling back to the tree view when any hash comparisons fails. This is equivalent to our *bottom-up shortcut*. Furthermore, when aligning child vertices, it does not take into account the cost of the recursive operations.

Bing Liu et al. have developed an effective algorithm for mining data records from Web pages [14]. The algorithm has two steps. In the first step it identifies the data region of the Web page and in the second one it extracts the records themselves. The algorithm works each time in a single page, so it does not compare the page trees. Although achieving good results, the algorithm only works with multi-record pages and therefore cannot be applied to on-line news pages, that are almost exclusively single-record pages.

Compared to the recent work in the literature, the work in this paper offers an alternative and uniform solution for three important problems in automatic Web data extraction: structure-based page classification, extractor generation, and data labeling. The fact that this solution is based on the well established concept of tree-edit distance brings the additional advantage of allowing the use of existing results for studying these problems from a new perspective.

# 7. CONCLUSIONS AND FUTURE WORK

In this paper we have introduced a new algorithm for calculating the edit distance between two given trees, which is based on a restricted form of top-down mapping. This algorithm, which we call *RTDM*, improves existing results in the literature [5, 25] for the problem of automatically analyzing the structure of Web pages.

Furthermore, we show how this algorithm can be applied to solve three important problems in automatic Web data extraction, namely: structure-based page classification, extractor generation, and data labeling. In particular, we have addressed the problem of automatically finding and fetching news available on Web sites, and extracting their components. Through experimentation with 35 news Web sites, we have demonstrated that the *RTDM* algorithm is highly effective for these tasks. Indeed, the results show an average of 87.71% correctly extracted news without any human intervention.

The approach provided by the RTDM algorithm is currently being used as the core of a fully operational Web news clipping sys-

tem, called AkwanClipping[3], which provides daily news from the most important Brazilian newspapers to over fifty companies.

As future work, we plan to generalize the proposed approach to deal with different application domains, especially those in which the schema of the data on the pages is complex. In fact, it is a challenge to provide a generic method for automatic Web data extraction [24]. Furthermore, we plan to use the *RDTM* algorithm to improve Web search engines by incorporating structural evidences derived from Web pages in addition to content evidences traditionally used by current search engines.

## Acknowledgements

This work is partially supported by project GERINDO (grant MCT/CNPq/CT-Info 552087/02-5) and by the fourth author's individual CNPq grant 304890/02-2.

## 8. REFERENCES

[1] A. Arasu, H. Garcia-Molina, and S. University. Extracting structured data from web pages. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 337–348. ACM Press, 2003.

[2] L. Arllota, V. Crescenzi, G. Mecca, and P. Merialdo. Automatic annotation of data extraction from large Web sites. In *Proceedings of the International Workshop on the Web and Databases*, pages 7–12, San Diego, USA, 2003.

[3] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, Harlow, England, 1st edition, 1999.

[4] V. Boyapati, K. Chevrier, A. Finkel, N. Glance, T. Pierce, R. Stockton, and C. Whitmer. Changedetector$^{TM}$: a site-level monitoring tool for the WWW. In *Proceedings of the 11th International Conference on World Wide Web*, pages 570–579. ACM Press, 2002.

[5] S. S. Chawathe. Comparing hierarchical data in external memory. In *Proceedings of the 25th International Conference on Very Large Data Bases*, pages 90–101, Edinburgh, Scotland, U.K., 1999.

[6] W. Chen. New algorithm for ordered tree-to-tree correction problem. *Journal of Algorithms*, 40:135–158, 2001.

[7] V. Crescenzi, G. Mecca, and P. Merialdo. RoadRunner: Towards automatic data extraction from large Web sites. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 109–118, Rome, Italy, 2001.

[8] V. Crescenzi, G. Mecca, and P. Merialdo. Wrapping-oriented classification of Web pages. In *Proceedings of the 2002 ACM Symposium on Applied Computing*, pages 1108–1112. ACM Press, 2002.

[9] D. Florescu, A. Levy, and A. Mendelzon. Database techniques for the world-wide web: a survey. *SIGMOD Rec.*, 27(3):59–74, 1998.

[10] M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. Xtract: a system for extracting document type descriptors from xml documents. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 165–176. ACM Press, 2000.

[11] S. Grumbach and G. Mecca. In search of the lost schema. In C. Beeri and P. Buneman, editors, *Proceedings of 7th International Conference on Database Theory*, Lecture Notes in Computer Science, pages 314–331, Jerusalem, Israel, 1999. Springer.

[12] A. Heydon and M. Najork. Mercator: A scalable, extensible web crawler. *World Wide Web*, 2(4):219–229, 1999.

[13] A. Laender, B. Ribeiro-Neto, A. Silva, and J. S. Teixeira. A brief survey of Web data extraction tools. *SIGMOD Record*, 31(2):84–93, 2002.

[14] B. Liu, R. Grossman, and Y. Zhai. Mining data records in web pages. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 601–606. ACM Press, 2003.

[15] J.-K. Min, J.-Y. Ahn, and C.-W. Chung. Efficient extraction of schemas for xml documents. *Information Processing Letters*, 85(1):7–12, 2003.

[16] A. Nierman and H. V. Jagadish. Evaluating structural similarity in XML documents. In *Proceedings of the 5th International Workshop on the Web and Databases (WebDB 2002)*, Madison, Wisconsin, USA, June 2002.

[17] S. M. Selkow. The tree-to-tree editing problem. *Information Processing Letters*, 6:184–186, Dec. 1977.

[18] K.-C. Tai. The tree-to-tree correction problem. *J. ACM*, 26(3):422–433, 1979.

[19] G. Valiente. An efficient bottom-up distance between trees. In *Proceedings of the 8th International Symposium on String Processing and Information Retrieval*, pages 212–219, Santiago, Chile, 2001. IEEE Computer Science Press.

[20] G. Valiente. Tree edit distance and common subtrees. Research Report LSI-02-20-R, Universitat Politècnica de Catalunya, Barcelona, Spain, 2002.

[21] J. T.-L. Wang, B. A. Shapiro, D. Shasha, K. Zhang, and K. M. Currey. An algorithm for finding the largest approximately common substructures of two trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):889–895, 1998.

[22] J. T. L. Wang and K. Zhang. Finding similar consensus between trees: an algorithm and a distance hierarchy. *Pattern Recognition*, 34:127–137, 2001.

[23] P. Willett. Recent trends in hierarchic document clustering: a critical review. *Information Processing and Management*, 24(5):577–597, 1988.

[24] G. Yang, I. V. Ramakrishnan, and M. Kifer. On the complexity of schema inference from web pages in the presence of nullable data attributes. In *Proceedings of the 12th International Conference on Information and Knowledge Management*, pages 224–231. ACM Press, 2003.

[25] W. Yang. Identifying syntactic differences between two programs. *Softw. Pract. Exper.*, 21(7):739–755, 1991.

[26] K. Zhang, D. Shasha, and J. T. L. Wang. Approximate tree matching in the presence of variable length don't cares. *J. Algorithms*, 16(1):33–66, 1994.

[27] K. Zhang, R. Statman, and D. Shasha. On the editing distance between unordered labeled trees. *Information Processing Letters*, 42(3):133–139, 1992.

[3]For more information see http://www.akwan.com.