A Fast and Robust Method for Web Page Template Detection and Removal

Karane Vieira[1] Altigran S. da Silva[1] Nick Pinto[1]
Edleno S. de Moura[1] João M. B. Cavalcanti[1] Juliana Freire[2]

[1]Universidade Federal do Amazonas
Departamento de Ciência da Computação
Manaus, AM, Brazil
{kmv,alti,ndyp,edleno,john}@dcc.ufam.edu.br

School of Computing
[2]University of Utah
Salt Lake City, UT, USA
juliana@cs.utah.edu

ABSTRACT

The widespread use of templates on the Web is considered harmful for two main reasons. Not only do they compromise the relevance judgment of many web IR and web mining methods such as clustering and classification, but they also negatively impact the performance and resource usage of tools that process web pages. In this paper we present a new method that efficiently and accurately removes templates found in collections of web pages. Our method works in two steps. First,the costly process of template detection is performed over a small set of sample pages. Then, the derived template is removed from the remaining pages in the collection. This leads to substantial performance gains when compared to previous approaches that combine template detection and removal. We show, through an experimental evaluation, that our approach is effective for identifying terms occurring in templates—obtaining F-measure values around 0.9, and that it also boosts the accuracy of web page clustering and classification methods.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms

Algorithms, Experimentation

Keywords

Web template extraction, Web page noise removal

1. INTRODUCTION

The availability of tools that simplify the design and implementation of data-intensive web sites has greatly contributed to the explosive growth of the Web. These tools involve "a combination of templates and design conventions" [12], including templates for common types and classes of pages, as well as sets of templates for common pages in sub-sites. By automatically populating these templates with content, web site designers and content producers of large web portals achieve high levels of productivity and improve the usability of the sites by enforcing the uniformity of the pages.

In a recent study, Gibson et al. [10] have found that templates represent between 40% and 50% of data on the Web and that this volume has been growing at a rate of approximately 6% per year. In addition, around 30% of the visible terms and hyperlinks appear in templates. The abundance of templates on the Web is considered harmful to many web mining and searching methods. Such methods usually base their judgment of relevance on the frequency and distribution of terms (words) and hyperlinks on web pages. Since templates contain a considerable number of common terms and hyperlinks which are replicated in large number of pages, a relevance assessment that does not take templates into account may turn out to be inaccurate, leading to incorrect results. This has already been demonstrated in pioneering works on template removal for web searching, page clustering and page classification [1, 21].

Templates can also negatively impact the performance of applications that manipulate web pages. Since they cause content to be replicated in several pages and occur in large volumes, processing and storing templates and their associated information is likely to lead to a waste of resources such as storage space, bandwidth and processing cycles. As the Web grows, this aspect must also be taken into consideration. Template detection and removal techniques follow the trends of recent work which aim to detect unimportant (or less important) information of web pages and web collections in an attempt to save computational resources and to speed up processing [8, 15, 2].

In this paper we present new method for detection and removal of templates that is both fast and accurate. The proposed method works in two steps. Initially, templates are detected using a set of sample pages. The patterns identified in the detection step are then used to remove the templates present in the other pages in the collection. This separation leads to an efficient process. Although the detection task can be costly, it is applied only to a small number of pages. On the other hand, the template removal, which has to be applied to a large number of pages, can be done through an inexpensive procedure.

Our procedures for detecting and removing templates are based on finding a mapping between the underlying tree structures of web pages. Based on this mapping, we detect identical nodes in the trees and subtrees that contain these nodes. Intuitively, when a given subtree that spans from the document root is detected in both input pages, we regard this subtree as a template. Once this subtree is found, it can be easily located and removed from other pages. The use of page structure for template detection leads to high precision even when a small number of sample pages is provided. This is in contrast to previous methods that either rely on the identification of frequent nodes [1] or consider only a node and its descendants [21]. These methods usually need to process a much large number of pages to reliably detect patterns. As we discuss in Section 6, we are able to obtain results of quality comparable to [21] using a very small percentage (between 5 and 10%) of the number of samples used in their experiments.

An important contribution of this paper is a new algorithm that finds optimal mappings between the DOM trees of web pages. This algorithm is based on a restricted formulation of the top-down mapping problem between two trees, which is particularly suitable for detecting structural similarities among web pages [7]. As we discuss in our experimental evaluation, this makes our method efficient and effective in practice.

To evaluate our approach, we have performed two sets of experiments. In the first set, we verify the effectiveness our method for detecting and removing templates. The results show that F-measure values aroung 0.9 are obtained using as few as 24 pages. In the second set of experiments, we verify the impact of our template removal procedure when used in conjunction with methods for clustering and classifying web pages. The results obtained are comparable to those obtained in [21] and show substantial improvement on the quality of these methods.

The remainder of this paper is organized as follows. In Section 2, we give an overview of the problem of template detection and review related work. We present a brief review on the main concepts related to tree mapping in Section 3. In Section 4, we introduce the new tree mapping algorithm, and the process of the template detection and removal method which uses this algorithm is described in Section 5. Our experimental evaluation and results are discussed in Section 6. We conclude in Section 7, where we outline directions for future research.

2. TEMPLATE DETECTION METHODS

The problems of template detection and web page cleaning have received considerable attention in the recent literature. This may be explained by the pressing demand for solutions that prevent templates from negatively impacting techniques for web searching and mining.

In their pioneer work on template detection, Bar-Yossef et al. [1] have provided a formal definition of the problem and discussed several shortcomings brought about by the widespread use of templates on the Web. They have proposed two algorithms which rely on the identification of identical *pagelets* [4] (i.e., topically coherent disjoint portions of web pages) occurring in a densely linked page collection. Experiments based on the Clever search engine using the ARC set of queries [3] have shown considerable improvement when one of the proposed algorithms is used, the *Local Template Detection Algorithm*. When a query is processed, this algorithm is combined with a strategy that filters out pages/pagelets that do not contain query terms in their text. Since the ranking algorithm used in Clever is based on link analysis only, this filtering strategy is crucial for obtaining accurate results.

Instead of pagelets, our approach based on the co-occurrence of subtrees in the pages. While pagelet identification is based on fixed heuristics, our approach is adaptive and it takes advantage of common structures found in the pages of a given web site. In addition, while our method addresses the removal of terms or words found in templates, Bar-Yossef et al. focused on pruning hyperlinks considered as useless to improve the results of Clever in finding authorities.

Yi et al.[21] proposed an alternative approach whose main goal is to find common *noisy* portions in the pages of a given web site. Their approach is based on a tree structure—the *Site Style Tree (SST)*, which is similar to a DOM tree. A SST summarizes the presentation styles and the contents found in a set of web pages. It is built by scanning a set of pages and adding to SST every distinct DOM node found. If the same (or similar) node is found in several pages, this is registered in the SST by means of a frequency counter. Once the SST is constructed, the likelihood of its nodes representing noisy nodes on the pages is evaluated based on the diversity of presentation styles and contents associated to it on the SST tree. Less diversity indicates there is a higher likelihood of noise. The final cleaning process, i.e., elimination of noisy nodes, is accomplished by mapping the DOM tree for each page into the SST. If a given DOM node of a page is mapped to a SST node considered as noisy, this node is removed. This process is applied to all pages in the target site. In contrast to the work reported in [1] and our work, the work in [21] does not directly address the problem of finding a template on a set of pages. Their focus is on finding common parts on web pages of a given site, with respect to formatting/style and contents and to evaluate the relative importance of these parts. However, their ultimate goal is the same, that is, removing from web pages content that may negatively impact searching and mining methods.

Similarly to [21], our approach relies on the underlying tree structure of the pages to be cleaned, but in a different way. We view a template as a subtree that is common to the DOM trees of the target web pages. Thus, we reduce the problem of template detection to the problem of finding a common subtree in a set of given trees. This simpler formulation works well in practice and, using a small number of sample pages, it is able to extract templates with high precision (see Section 6). In contrast, Yi et al. need to examine a large number of pages to accumulate reliable statistics. In their experiments, to obtain results with quality comparable to ours, they use over an order of magnitude more sample pages.

In addition, in their approach, two passes over the page collection are required. In the first pass, the SST is built and in the second, each page in the collection is compared against the SST. On the other hand, our approach requires a single pass over the collection: templates are first detected from sample pages and then they are removed from the remaining pages using an inexpensive procedure. Another advantage of our approach is that it is completely automatic. Yi et al., in contrast, require users to define thresholds that indicate whether elements in the SST are noisy or not and to tune the similarity comparison between DOM trees elements.

Debnath et al. [9] proposed an approach similar to the one in [21]. They also select portions of web pages, called *blocks*, which have an importance level above a given threshold. However, while Yi et al. defined the notion of importance in [21] based on features of the whole target site summarized in style trees, Debnath et al. estimate the importance for each individual block. Two distinct strategies are proposed. One is based on occurrence of similar blocks among the pages of the site and another is based on a specific predefined set of desired features that must be present on blocks. By using a set of parametrized heuristics to delimit blocks, identify features and calculate importance, the method achieved good precision and recall levels when applied to collections of news pages from several web sites.

Song et al. [16] proposed a learning method that automatically assigns importance weights to hierarchically arranged segments in web pages, termed as *blocks*. The method requires blocks in sample web pages to be labeled by users based on their judgment of the importance of each block. Next, using the labeled blocks as training data, classifiers are used to automatically label other unseen pages. Template identification can be seen as a specific instance of this problem, since templates can be regarded as sets of blocks that have very low importance. However, as we show in this paper, templates can be automatically (and reliably) detected without any training.

Using a method based on the approaches in [1]and [21], with some simplifications, Gibson et al. [10] have conducted an extensive survey on the use of templates on the Web which revealed interesting facts. For instance, they have found that templates currently represent between 40% and 50% of the volume of the Web and that this volume is growing at a rate of approximately 6% per year. In addition, around 30% of the visible terms and hyperlinks appear in templates.

3. TREE MAPPING

To detect templates in a collection of web pages, our method tries to identify mappings among their tree structure. In what follows, we review the concept of tree mapping and describe a restricted version of the problem used in our approach.

We represent each HTML page as a *labeled ordered rooted tree* that corresponds to its underlying DOM tree. From now on, we refer to labeled ordered rooted trees simply by trees, except when explicitly stated.

The concept of tree mapping was introduced in [17] and is formally defined as follows.

Definition 1. *(Tree Mapping) Let T_x be a tree and let $t_x[i]$ be the i-th vertex of tree T_x in a preorder walk. 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 1 illustrates a mapping between two trees.

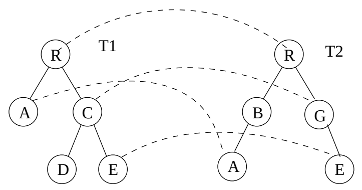

Figure 1: Mapping between two labeled ordered rooted trees.

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. Tree operations are commonly considered: (a) vertex removal, (b) vertex insertion, and (c) vertex replacement. In Figure 1, a dashed 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 dashed lines should be deleted, and vertices of T_2 not touched should be inserted. If we associate a *cost* to each of the operations considered, it's possible to evaluate the cost of a given mapping between two trees, as defined below.

Definition 2. *(Mapping Cost) Let M be a mapping between tree T_1 and tree T_2; S a subset of pairs $(i, j) \in M$ with distinct labels; D the set of nodes in T_1 that do not occur in any $(i, j) \in M$; and I the set of nodes in T_2 that do not occur in any $(i, j) \in M$. The cost of mapping M is given by $c = |S|p + |I|q + |D|r$, 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, although specific applications may require the assignment of distinct costs to each type of operation. Other applications may even require a distinct set of operations.

As many mappings can exist between two trees, an important problem is how to find the optimal mapping, that is, the one with the minimal cost among all possible mappings. This problem is also known as the *Tree-Edit Distance(TED)* problem [14, 17]. From now on, we refer to the optimal tree mapping problem simply as the mapping problem.

Despite the inherent complexity of the mapping problem in its generic formulation [6, 22], there are several practical applications that can be modeled 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 [18, 19].

Our approach is based on the top-down mapping formulation, that restricts the removal and insertion operations to take place only in the leaves of the trees. Figure 2(a) illustrates a top-down mapping which is formally defined as follows.

Definition 3. *A mapping M between a tree T_1 and a tree T_2 is said to be* top-down *if and only if for every pair $(i_1, i_2) \in M$ there is also a pair (parent(i_1),parent(i_2)) $\in M$.*

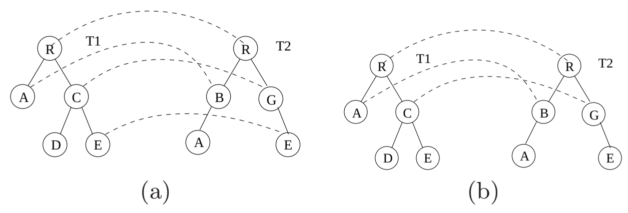

Figure 2: Examples of generic (a) and restricted (b) top-down mappings.

The first algorithm for the top-down edit distance problem was proposed by Selkow [14]. One of the most popular algorithms for the problem is presented in [5] and has time complexity of $O(n_1n_2)$, where n_1 and n_2 correspond to the number of nodes in the trees being mapped. Top-down mappings have been successfully applied to several web related applications such as document categorization. For instance, Nierman and Jagadish [13] use a top-down distance algorithm to cluster XML documents.

In our work we use a restricted form of top-down mapping between two pages in which, 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 4. *A top-down mapping M between a tree T_1 and a tree T_2 is* restricted *if and 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 descendant of i_1 or i_2 in M.*

Figure 2(b) shows a restricted top-down mapping. Similar to 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 concept of restricted top-down mapping was introduced in [7] along with the *RTDM* algorithm. RTDM is well suited for problems that require the evaluation of structural similarity between web pages. Because edit operations are restricted to leaf nodes, while a mapping is constructed, there is no need to look for nodes in subtrees rooted at a changed node since they cannot appear in the mapping. As a result, only nodes in common subtrees spanning from the root to the leaves may be considered for the mapping. As we discuss later, we regard these subtrees as indicators of the presence of a template.

4. THE RTDM-TD ALGORITHM

Whereas the main goal of the RTDM algorithm is to determine the cost of the minimal mapping, i.e., the tree-edit distance between two input trees, to perform template detection we need to find the actual sequence of operations that lead to this minimal mapping. Here, we introduce *RTDM-TD*, an extension of RTDM in which the mapping result is built while the mapping is determined. There are several differences between the two algorithms. RTDM requires as input a threshold, since processing can be halted in cases in which the partial cost being calculated is already above this threshold. As RTDM-TD needs to construct the complete sequence of operations, regardless of the cost, it does not need such a threshold. Another distinction is that RTDM-TD treats certain nodes in a special way. Because nodes that are identical in two trees are regarded as being part of a template, RTDM-TD keeps track of cases in which no insertion, removal or update operations were applied to a given node.

The RTDM-TD algorithm, shown in Figure 3, is an adaptation of the algorithms by Yang [20] and Reis [7] applied to obtain minimal restricted top-down mappings. It takes as input two trees and outputs two matrices: the *cost* matrix M; and the *backtracking* matrix B. Elements in matrix M represent the minimum cost for mapping a subtree of T_1 to a subtree of T_2. In matrix B, each element $B[i, j]$ represents the operations over two nodes, $t_1[i]$ of T_1 and $t_2[j]$ of T_2, that lead to the minimum cost.

Each $B[i, j]$ is a tuple of the form $\langle op, src, cost, next\rangle$, where *op* stores the operation that led to minimum cost, *src* indicates the node in T_1 that was used in this operation, *cost* stores the cost of this operation and *next* is a pointer for the backtracking matrix corresponding to the children of t_1 and t_2. As we explain later, the backtracking matrices are used for retrieving the nodes composing the template. Notice that we do not need to represent the value of the cost in component *cost*, since it is already available in M—we do this only for convenience in the retrieval process. Also, notice that we only keep track of nodes of T_1 in *src*, since for template detection, we are only interested in nodes that appear in the two trees (the corresponding node in T_2 will be the same).

In the RTDM-TD algorithm, functions *replace*, *delete* and *insert* give the costs of vertex replacement, vertex removal and vertex insertion, respectively. Notation $t_x[k]$ is used to refer to the k_{th} node of tree T_x according to a post-order traversal and $T_x[k]$ refers to the subtree of T_x rooted at $t_x[k]$. Moreover, $\delta(T_x, i)$ is used to refer the post-order traversal index of the i_{th} child of the root of the tree T_x.

As other top-down mapping algorithms [20, 7], RTDM-TD works similarly to algorithms that solve string edit distance. In our case, we are trying to find the best assignment between the children of the roots of T_1 and T_2. Thus, when both of these children nodes have their own children nodes, we make a recursive call to solve the mapping in lower levels of the trees (Line 28-29). In the algorithm, this only occurs when processing two children nodes that have the same label (Line 25) but are not roots of identical subtrees (Line 27).

In our implementation, the *ident* function used in Line 25 consists only of looking into sets of equivalent subtrees. If two sub-tress are in the same set, they are considered identical. These sets are computed from the two input trees in a pre-processing step, similar to what is done in [18] and [7]. As in the original RTDM algorithm, in RTDM-TD this approach is applicable because we only look for the identical subtrees of the *same level.* This pre-processing has linear cost with respect to the number of vertices in the trees.

After the recursive call in Lines 28-29, argument B' contains a "reference" to the backtracking matrix corresponding to the children of the nodes being currently processed. It is assigned to $B[i, j].next$ in Line 44.

The traditional top-down mapping algorithm by Chawathe [5] has time-complexity of $O(n_1n_2)$, where n_x is the size of T_x. For all cases, $O(n_1n_2)$ is the best, the expected and the worst cases. The RTDM-TD algorithm also has a worst case complexity of $O(n_1n_2)$, 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-TD algorithm occurs when the two trees being compared are all identical, except for their leaves. In all other cases, the cost

```
1  **RTDM-TD**($T_1$, $T_2$, $B$)
2  begin
3    let $m$ be the number of children of $T_1$ root
4    let $n$ be the number of children of $T_2$ root
5    for $i = 1$ to $m$
6        $C_i \leftarrow descendants(\delta(T_1, i))$
7        $M[i, 0] \leftarrow M[i-1, 0] + \sum_k^{t_1[k] \in C_i} delete(t_1[k])$
8    end
10   for $j = 1$ to $n$
11       $C_j \leftarrow descendants(\delta(T_2, j))$
12       $M[0, j] \leftarrow M[0, j-1] + \sum_k^{t_2[k] \in C_j} insert(t_2[k])$
13   end
14   for $i = 1$ to $m$
15       for $j = 1$ to $n$
16           $B' \leftarrow null$
17           $C_i \leftarrow descendants(\delta(T_1, i))$
18           $C_j \leftarrow descendants(\delta(T_2, j))$
19           $D \leftarrow M[i-1, j] + \sum_k^{t_1[k] \in C_i} delete(t_1[k]) +$
20                   $delete(\delta(T_1, i))$
21           $I \leftarrow M[i, j-1] + \sum_k^{t_2[k] \in C_j} insert(t_2[k]) +$
22                   $insert(\delta(T_2, j))$
23           $S \leftarrow M[i-1, j-1]$
24           $N \leftarrow \infty$
25           if $ident(T_1[\delta(T_1, i)], T_2[\delta(T_2, j)])$
26               $S \leftarrow S + 0$
27           elsif $t_1[\delta(T_1, i)] = t_2[\delta(T_2, j)]$
28                   $N \leftarrow S +$ **RTDM-TD**($T_1[\delta(T_1, i)]$,
29                                               $T_2[\delta(T_2, j)]$,$B'$)
30           else
31                   $S \leftarrow replace(t_1[\delta(T_1, i)], t_2[\delta(T_2, j)])$
32                   if $\delta(T_1, i)$ is a leaf
33                       $S \leftarrow S + \sum_k^{t_2[k] \in C_j} insert(t_2[k])$
34                   elsif $\delta(T_2, j)$ is a leaf
35                       $S \leftarrow S + \sum_k^{t_1[k] \in C_i} delete(t_1[k])$
36                   fi
37           fi
38           $M[i, j] \leftarrow min(D, I, S, N)$;
39           $B[i, j].cost \leftarrow M[i, j]$;
40           $B[i, j].src \leftarrow t_1[\delta(T_1, i)]$;
41           $B[i, j].next \leftarrow B'$;
42           if  $M[i, j] = D$  $B[i, j].op \leftarrow$ "delete"
43           elsif  $M[i, j] = I$  $B[i, j].op \leftarrow$ "insert"
44           elsif  $M[i, j] = S$  $B[i, j].op \leftarrow$ "update"
45           elsif  $M[i, j] = N$  $B[i, j].op \leftarrow$ "no_op"
46           fi
47       end
48   end
49   return $M[m, n]$
50 end
```

Figure 3: The *RTDM-TD* Algorithm.

is amortized by the short-cuts in Lines 16, which we call the *bottom-up short-cut*, or in Line 21, which we call the *top-down short-cut*. Notice that this important feature is inherited from the original RTDM algorithm.

After the execution of RTDM-TD, we execute the *retrieveTemplate* algorithm, described in Figure 4. This algorithm receives as input the backtracking matrix B, returned by the RTDM-TD, and returns the set of nodes that compose a template. We begin by interacting from the last cell of B, traversing in the matrix according to the operations found, until the first line or the first column is reached. Every time we find an *update* operation with a $B[i, j].cost$ that has not changed, the node $B[i, j].src$ is added to the *template* set. The same is done when we find that no operation was performed. However, in this last case, we make a recursive call to retrieve the template nodes that are children of the current node $B[i, j].src$, using the pointer $B[i, j].next$ (Line 25).

The RTDM-TD algorithm and retrieveTemplate are used as the basis of our template detection and removal method, described in the next section.

```
1  **retrieveTemplate**($B$)
2  begin
3    let $m$ number of rows of $B$
4    let $n$ number of columns of $B$
8    $i \leftarrow m$
9    $j \leftarrow n$
10   while  $i > 0$ and $j > 0$
12     if $B[i, j].op =$ "delete"
13       $j \leftarrow j - 1$
14     elsif $B[i, j].op =$ "insert"
15       $i \leftarrow i - 1$
16     elsif ($B[i, j].op =$ "update" and
18             $B[i, j].cost = B[i-1, j-1].cost$)
19       $template \leftarrow template \cup \{B[i, j].src\}$
20       $template \leftarrow template \cup$ descendants($B[i, j].src$)
21       $i \leftarrow i - 1$
22       $j \leftarrow j - 1$
23     elsif $B[i, j].op =$ "no_op"
24       $template \leftarrow template \cup \{B[i, j].src\}$
25       **retrieveTemplate**($B[i, j].next$)
26       $i \leftarrow i - 1$
27       $j \leftarrow j - 1$
28     fi
29   end
30   return $template$
31   end
```

Figure 4: The *retrieveTemplate* Algorithm.

5. TEMPLATE DETECTION AND REMOVAL

The motivation for the design of RTDM-TD came from the observation that templates are just fragments of HTML code included in an collection of HTML documents and they are fairly regular on specific portions of a site. Templates are usually automatically generated by programs and are copied to multiple pages. In fact, two important reasons for the widespread use of are to ensure uniformity and to speed up development.

This means, in practice, that a template can be viewed as a subtree that is common to the DOM trees of a a web page collection. Thus, we reduce the problem of template detection to the problem of finding a subtree common to a set of given trees. This simple formulation works in practice and extracts templates with high precision, as shown by our experiments presented in Section 6.

In addition to its simplicity and precision, our method has a unique feature that represents an improvement over previous methods presented in the literature. It takes advantage of the inherent regularity of templates and is able to detect them without having to examine a large number of pages. This is also demonstrated by our experiments in Section 6.

To find the common subtree that represents the template in a set of pages, we compare these pages using the mapping algorithm presented in Section 3. This procedure is detailed in the FindTemplate algorithm, shown in Figure 5.

Before discussing the FindTemplate algorithm, we first discuss the ExtractSubTree algorithm, presented in Figure 6.

```
1  FindTemplate($W$)
2  begin
3    let $W$ be a set of pages from a given web site
4    $P_x \leftarrow RandomPick(W)$
5    $P_y \leftarrow RandomPick(W)$
6    $S_0 \leftarrow ExtractSubTree(P_x, P_y)$
8    for $i \leftarrow 1$ to $MaxPages - 2$
9        $P \leftarrow RandomPick(W)$
10       if $Match(S_{i-1}, P)$
11         $S_i \leftarrow S_{i-1}$
12       else $S_i \leftarrow ExtractSubTree(S_{i-1}, P)$
13       fi
14   end
15   return $S_i$
16 end
```

Figure 5: The *FindTemplate* Algorithm.

```
1  ExtractSubTree($T_x$,$T_y$)
2  let $T_x$ and $T_y$ be trees
3  begin
4    RTDM-TD($T_x$, $T_y$, $B$)
5    $template\_nodes \leftarrow$ retrieveTemplate($B$)
6    $S \leftarrow$ the smallest $T_x$ subtree
7           that contains all nodes in $template\_nodes$
8    return $S$
9  end
```

Figure 6: The *ExtractSubTree* Algorithm.

It simply applies the RTDM-TD algorithm (Figure 3) to obtain a mapping between two given trees T_x and T_y followed by the application of retrieveTemplate (Figure 4) to obtain the nodes that compose a template. The resulting common subtree is the smallest one that contains all of these common nodes.

FindTemplate algorithm begins by applying the RandomPick function to randomly select and remove two of the pages from the input set of web pages W (Lines 4–5). Next, the ExtractSubTree algorithm is used to extract a common subtree between these pages. At each iteration of the algorithm (Lines 8–14), a a subtree previously generated (S_{i-1}) is matched against a newly selected web page P from W. If P does not contain S_{i-1}, then a new subtree is extracted (Line 12). Finally, the last subtree extracted is returned as a result.

An important aspect related to the FindTemplate is the number of iterations it requires in the loop of Lines 8–14 to obtain the correct template. As discussed above, templates are fairly regular and they are present in all of the pages. However, as between any two randomly picked pages there can be a common subtree larger than the template, we usually need more pages to converge to the correct template. We represent the number of pages necessary to be examined by $MaxPages$. In our experiments we have found that a few dozen of pages are usually enough.

If we consider that all pages have an average of n nodes, the ExtractSubTree has a complexity of $O(n^2)$. Then, if $MaxPages$ is equal to a constant M, FindTemplate has a worst-case complexity of $O(n^2)$. Notice that, in practice, we expect Match(S_{i-1},P) to have complexity $O(n)$ for many cases.

In contrast to previous approaches proposed in the literature, our method does not need to examine a large quantity of pages to find the correct template. This is important for applications in which not all pages are known in advance, for instance, during a web crawling process or while processing a stream of pages. Once a template has been detected, the removal process from a given page consists in finding a subtree in this page, what is obtained in linear time. We omit the description of the procedure to remove the template due to its simplicity.

6. EXPERIMENTAL RESULTS

This section presents experiments we have performed to verify the effectiveness of our approach. First, to verify the effective our method for the tasks of detecting and removing templates, we manually extracted sets of terms from templates and compared them with the sets of terms present in the templates automatically identified by our template detection procedure. We then study the impact of our template removal procedure when used conjunction with clustering and classification methods, similar to the evaluation done in [21]. The experiments and their results are presented and discussed below.

6.1 Direct Experimental Evaluation

We selected 10 (real) web sites. For each site i, we manually identified the template by visually inspecting the pages. We built a reference set S_i containing the terms (words) present in the template. Then, for each of these sites we applied our method to automatically remove the template and generated a corresponding set T_i of the terms present in the detected template. Sets S_i and T_i were then compared using the well-known F-measure defined as: $F_i = 2(R_i.P_i)/(R_i + P_i)$, where $R_i = |S_i \cap T_i|/|S_i|$ (Recall) and $P_i = |S_i \cap T_i|/|T_i|$ (Precision).

Five of the sites, CNET, J&R, PCConnection (PC) and PCMagazine and ZDNET, are e-commerce sites used in the experiments previously presented in [21]. The other five are popular sites in distinct domains: CNN, E-Jazz, Encyclopedia Mythica, UBL (Ultimate Band List) and Wikipedia.

In Figure 7, we show F-measure values obtained by running our methods using a number of pages varying from 2 to 100, for each of the web sites considered.

We notice that for all but one site, E-Jazz, F-measure values above 0.95 were achieved when the number of sample pages is fewer than 25. Even for e-jazz, the F-measure value with this number of pages is above 0.85. This indicates that a suitable approximation of the template was found, since the terms present in the templates were precisely identified. Also, notice that the number of samples pages required for this to happen is rather small.

Being able to identify the correct set of terms present in the templates is important to neutralize the negative effects caused by templates in web searching and web mining methods. In the next section we show experiments in which we evaluate the impact of using our template detection technique as a preprocessing step in some of such methods.

6.2 Impact on Web Mining Problems

Below we evaluate the impact of our method on web mining tasks. These experiments are the same as those reported in [21], but with slightly distinct data sets. As in [21], we have used several pages containing information on products from five distinct categories: *Notebook*, *Digital Camera*, *Mobile Phone*, *Printer* and *TV*. These pages were collected from

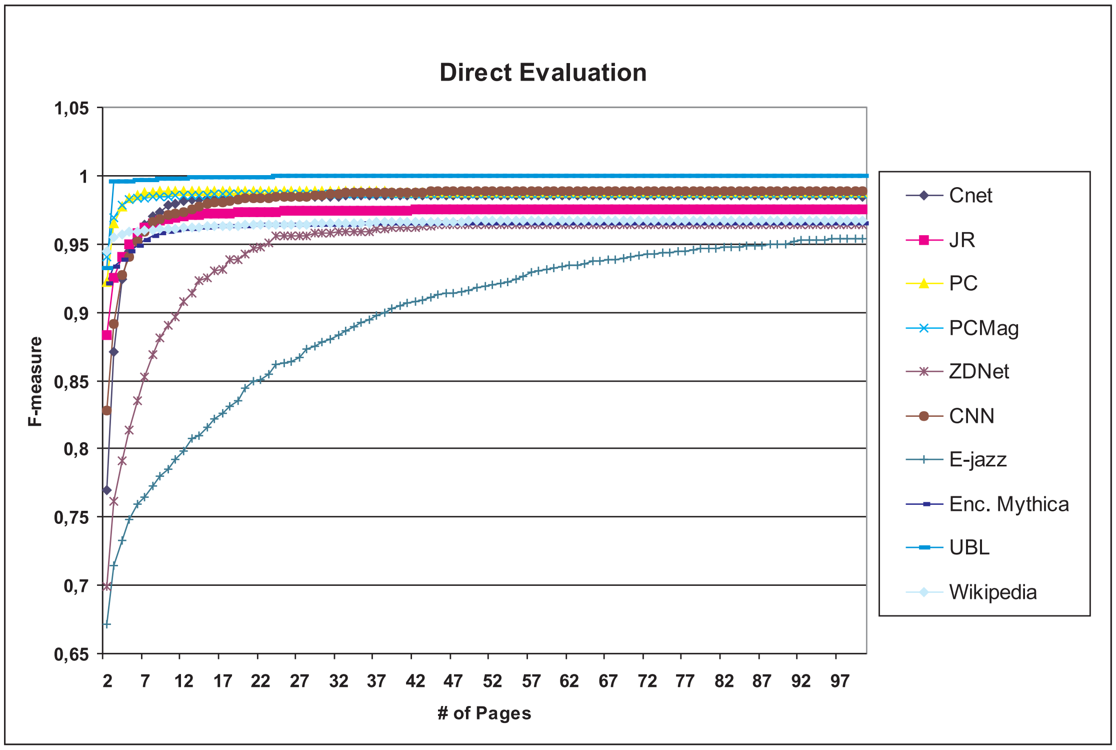

Figure 7: Direct evaluation of template removal based on terms.

five distinct commercial web sites: PCConnection (PC), CNet, PCMag, J&R and ZDnet. In Table 1, we show the number of pages collected in each category from each site.

Sites	PC	CNet	J&R	PCMag	ZDNet
Notebook	600	497	74	150	218
Camera	169	227	216	143	132
Mobile	8	120	47	52	118
Printer	496	511	133	117	97
TV	278	451	161	103	141

Table 1: Number of pages in each class per web site.

There are two noteworthy distinctions between the datasets used in [21] and the ones we have used here. First, we use pages from the PCConnection web site instead of the pages from the Amazon web site used in [21]. We had to do this because of difficulties experimented when crawling this site, possibly due to some form of spider trap deployed there. Second, even for the remaining sites, the pages we collected have a format that is distinct from the ones used in [21], since these design of the pages have passed through many changes over the years.

Despite these distinctions, the pages in our datasets are similar in contents and number of pages to those used in [21]. More importantly, the pages we use also present navigational information, advertisements, and other information not directly related to the main topic of the pages. This information is located on automatically generated templates and, as demonstrated by the results we present, is correctly removed along with the templates detected by our method.

6.2.1 *Impact on Clustering*

To evaluate the impact of using our template detection method on a clustering task, we use the K-means algorithm to build clusters of pages of a same category of products taking as input a set of pages composed by all pages we have collected as explained above.

The experiments consisted of running our implementation of K-means 800 times with randomly selected seeds. To evaluate the clusters resulting from each run, we used the F-measure, which corresponds the harmonic mean of the precision and recall metrics. The computation of precision and recall was accomplished by taking the five categories of the pages as a reference.

Notice that we have applied the same experimental procedure adopted in [21] to our page set, which, as we have already mentioned, has the same features as the page set used in that paper.

Figure 8 shows the number of experimental runs that have achieved an F-measure value that falls on one of the value intervals $[0.0, 0.1), [0.1, 0.2), \ldots, [0.9, 1.0]$. Each bar correspond to the average F value of the 5 categories, before and after the application of the template removal method. Similar results were presented in [21].

The bars in Figure 8 clearly show the positive impact caused by our method over the clustering task. For instance, it can be observed that 40% of the clustering experiments over the set of cleaned pages reached F-measure values that were not achieved by any of the experiments over the set of original (noisy) pages.

In Table 2 we offer a perspective of the impact caused by our method on a clustering task in comparison with the impact caused by the method proposed in [21]. In the first row, we reproduce the data reported in that paper for F-measure

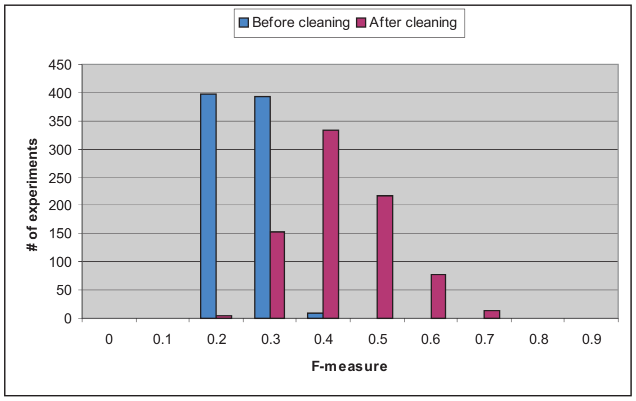

Figure 8: Average results of the clustering tasks before and after template removal.

values obtained with the clustering experiments before and after applying the cleaning method based on Site Style Tress (SST). Based on these values, we calculated the percentage gain in F-measure. Similarly, in the second row we show the same figures obtained with our proposed method. We notice that, in absolute values, the F-measure values reported in [21] are superior, after and before cleaning, that the values obtained in our method. However, the percentage gains are comparable, with a slight advantage to our method.

Method	Original Pages (F)	Cleaned Pages (F)	Gain
SST	0,506	0,751	48,42%
TDR	0,306	0,482	57,52%

Table 2: Impact of different cleaning techniques for clustering web pages.

We stress that the experimental procedure we use is the same as in [21], but the sets of pages used in the two experiments, although similar, are not the same. This explains the difference in absolute F-measure values for the uncleaned pages. For the same reason, it is not possible to claim that our method is superior, considering the impact on clustering. We can, however, conclude that the two methods lead to comparable impact in such a task.

6.2.2 *Impact on Classification*

The evaluation of impact our method over classification also follows the experimental procedure proposed in [21]. Using a Naïve Bayes classifier [11], we experiment with three different configurations for classification tasks using all possible pairs of product categories. The configurations are summarized in Table 3. For each of them, we first train the Naive Bayes classifier using a Training Set (TR) and then run the classifier over the corresponding Test Set (TE).

Tables 4 and 5 present results of the classification experiments for each configuration over all pairs of categories before and after cleaning, using F-measure and Accuracy respectively. All results show a noticeable improvement on the quality of the classification process.

Table 6 provides a comparison of the relative impact on classification tasks caused by our method (TDR) and the impact caused by the method proposed in [21] (SST). For this comparison we computed the gain obtained in terms of F-measure (F) and in terms of Accuracy (A). All figures in

Conf.	Training Set (TR)	Test Set (TE)
1	Pages from categories p and q from site i	Pages from categories p and q from all sites except i
2	Pages from category p from site i and pages from category q from site $j \neq i$	Pages from categories p and q from all sites except i and j
3	Pages from category p from site i and pages from category q from site $j \neq i$	Pages from categories p and q from all sites except pages in the training Set

Table 3: Configuration of classification experiments

SST were calculated from F-measure and accuracy values provided in [21]. Notice that the relative gain in Configurations 1 and 2 are slightly better in our method, while in Configurations 3 the SST method reaches slightly better gain levels. This leads us to conclude that the methods cause a comparable impact on classification task, as they do for clustering tasks.

	Conf. 1		Conf. 2		Conf. 3	
	TDR	SST	TDR	SST	TDR	SST
F	11,17%	6,84%	45,51%	26,80%	52,30%	58,36%
A	10,03%	6,33%	32,85%	23,00%	52,64%	41,02%

Table 6: Impact of different cleaning techniques for web page classification.

The results in Tables 4, 5 and 6 were all obtained using 100 pages for detecting the template. However, similar results are obtained when we use fewer pages for the detection. To demonstrate this, we show in Figure 9 values of F-measure for the classification process in all three configurations, considering that N = 10,20,30,40 and 50 pages were used for detecting the template. The results with 100 are also shown as a reference.

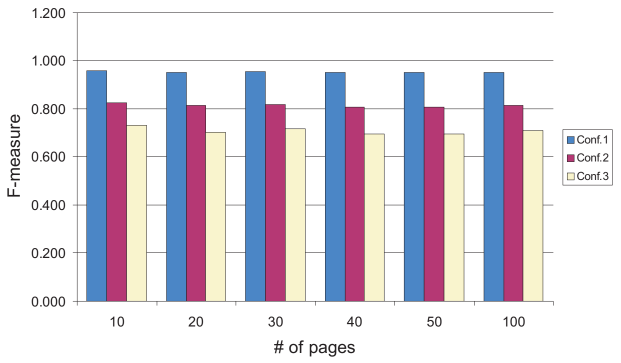

Figure 9: Results of classification for all three configurations after template removal. Templates were detected using N = 10,20,30,40,50 and 100 pages.

We notice that the F-measure values are identical for all values of N, thus the same gain values presented in Table 6 are also observed with all these values. This corroborates our results in Section 6.1 and confirms that useful templates can be obtained with only a small fraction of the pages.

Categories		Conf. 1		Conf. 2		Conf. 3	
p	*q*	Orig.	Clean.	Orig.	Clean.	Orig.	Clean.
Notebook	Camera	0.902	0.978	0.583	0.813	0.417	0.625
Notebook	Mobile	0.762	0.845	0.347	0.615	0.251	0.462
Notebook	Printer	0.902	0.989	0.574	0.776	0.419	0.599
Notebook	TV	0.902	0.987	0.532	0.747	0.382	0.584
Camera	Mobile	0.789	0.901	0.389	0.736	0.293	0.568
Camera	Printer	0.866	0.961	0.636	0.818	0.477	0.643
Camera	TV	0.911	0.975	0.579	0.816	0.427	0.623
Mobile	Printer	0.822	0.875	0.581	0.759	0.439	0.573
Mobile	TV	0.675	0.885	0.506	0.742	0.382	0.556
Printer	TV	0.911	0.989	0.527	0.823	0.379	0.655
Average F		**0.844**	**0.939**	**0.525**	**0.765**	**0.387**	**0.589**

Table 4: F-measure classification results for each configuration over all pairs of categories before and after cleaning.

Categories		Conf. 1		Conf. 2		Conf. 3	
p	*q*	Orig.	Clean.	Orig.	Clean.	Orig.	Clean.
Notebook	Camera	0.906	0.980	0.611	0.817	0.438	0.631
Notebook	Mobile	0.795	0.860	0.409	0.636	0.288	0.488
Notebook	Printer	0.903	0.989	0.621	0.788	0.452	0.612
Notebook	TV	0.903	0.988	0.578	0.755	0.411	0.598
Camera	Mobile	0.798	0.905	0.459	0.751	0.336	0.587
Camera	Printer	0.880	0.962	0.703	0.836	0.531	0.668
Camera	TV	0.913	0.975	0.625	0.823	0.462	0.632
Mobile	Printer	0.842	0.889	0.679	0.782	0.530	0.613
Mobile	TV	0.719	0.896	0.608	0.762	0.457	0.582
Printer	TV	0.914	0.989	0.561	0.827	0.403	0.664
Average Accuracy		**0.857**	**0.943**	**0.585**	**0.778**	**0.431**	**0.608**

Table 5: Accuracy classification results for each configuration over all pairs of categories before and after cleaning.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a new approach to the problem of detecting and removing templates from web pages. In our approach, templates are detected by constructing mappings between the DOM trees of distinct pages and finding subtrees that are common in these pages. We showed that, not only can this mapping be computed efficiently using our RTDM-TD algorithm, but also, high precision can be obtained with a small number of samples. In addition, once a template is detected, it can be removed from new web pages by a simple (and inexpensive) procedure. Our experimental results also show that our approach leads to substantial improvements in quality for both clustering and classification tasks.

We notice that our approach can be extended to derive multiple templates. Such an extension is planed as future work.

Another interesting direction of future work we intend to pursue is to apply our work to search engines. The simplicity and efficiency of our approach make it especially suitable for this task—few samples are needed for template detection and a single pass over the Web pages is sufficient for template removal.

Acknowledgments

This work is partially supported by projects GERINDO (CNPq/CT-INFO 552.087/02-5), SIRIAA (CNPq/CT-Amazônia 55.3126/2005-9), 5S-VQ (CNPq/CT-INFO 55.1013/2005-2), and by individual grants from CNPq to Altigran S. da Silva (303032/2004-9), Edleno S. de Moura (303576/2004-9), João M. B. Cavalcanti (303738/2006-5) and by CAPES to Karane Mariano. Juliana Freire is supported by the National Science Foundation (grants IIS-0534628, OCE-0424602 ,IIS-0534628, IIS-0513692, and CNS-0528201) and by a University of Utah Seed Grant.

8. REFERENCES

[1] Z. Bar-Yossef and S. Rajagopalan. Template detection via data mining and its applications. In *Proceedings of the International Conference on the World Wide Web*, pages 580–591, 2002.

[2] K. Bharat, A. Z. Broder, J. Dean, and M. R. Henzinger. A comparison of techniques to find mirrored hosts on the WWW. *Journal of the American Society for Information Science*, 51(12):1114–1122, 2000.

[3] S. Chakrabarti, B. Dom, P. Raghavan, S. Rajagopalan, D. Gibson, and J. M. Kleinberg. Automatic resource compilation by analyzing hyperlink structure and associated text. *Computer Networks*, 30(1-7):65–74, 1998.

[4] S. Chakrabarti, M. Joshi, and V. Tawde. Enhanced topic distillation using text, markup tags, and hyperlinks. In *Proceedings the of ACM Conference on Research and Development in Information Retrieval*, pages 208–216, 2001.

[5] S. S. Chawathe. Comparing hierarchical data in external memory. In *Proceedings of the Very Large Data Bases Conference*, pages 90–101, 1999.

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

[7] D. de Castro Reis, P. B. Golgher, A. S. da Silva, and A. H. F. Laender. Automatic web news extraction using tree edit distance. In *Proceedings of the International Conference on the World Wide Web*, pages 502–511, 2004.

[8] E. S. de Moura, C. F. dos Santos, D. R. Fernandes, A. S. da Silva, P. Calado, and M. A. Nascimento. Improving web search efficiency via a locality based static pruning method. In *Proceedings of the International Conference on the World Wide Web*, pages 235–244, 2005.

[9] S. Debnath, P. Mitra, and C. L. Giles. Automatic extraction of informative blocks from webpages. In *ACM Symposium on Applied Computing*, pages 1722–1726, 2005.

[10] D. Gibson, K. Punera, and A. Tomkins. The volume and evolution of web page templates. In *Proceedings of the international conference on the World Wide Web - Poster Session*, pages 830–839, 2005.

[11] T. M. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[12] J. Nielsen. User interface directions for the web. *Communications of the ACM*, 42(1):65–72, 1999.

[13] A. Nierman and H. V. Jagadish. Evaluating structural similarity in XML documents. In *Proceedings of the International Workshop on the Web and Databases*, June 2002.

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

[15] A. Soffer, D. Carmel, D. Cohen, R. Fagin, E. Farchi, M. Herscovici, and Y. S. Maarek. Static index pruning for information retrieval systems. In *Proceedings the of ACM Conference on Research and Development in Information Retrieval*, pages 43–50, 2001.

[16] R. Song, H. Liu, J.-R. Wen, and W.-Y. Ma. Learning block importance models for web pages. In *Proceedings of the International Conference on the World Wide Web*, pages 203–211, 2004.

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

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

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

[20] W. Yang. Identifying syntactic differences between two programs. *Software – Practice And Experience*, 21(7):739–755, 1991.

[21] L. Yi, B. Liu, and X. Li. Eliminating noisy information in web pages for data mining. In *Proceedings of the International ACM Conference on Knowledge Discovery and Data Mining*, pages 296–305, 2003.

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