A NEW FREQUENT SIMILAR TREE ALGORITHM
MOTIVATED BY DOM MINING
Using RTDM and its New Variant — SiSTeR

Barkol Omer, Bergman Ruth and Golan Shahar
HP Labs, Technion City, Haifa, Israel

Keywords: Frequent trees, Tree edit distance, RTDM, DOM, Web mining, Web data records.

Abstract: The importance of recognizing repeating structures in web applications has generated a large body of work on algorithms for mining the HTML Document Object Model (DOM). A restricted tree edit distance metric, called the Restricted Top Down Metric (RTDM), is most suitable for DOM mining as well as less computationally expensive than the general tree edit distance. Given two trees with input size \( n_1 \) and \( n_2 \), the current methods take time \( O(n_1 \cdot n_2) \) to compute RTDM. Consider, however, looking for patterns that form subtrees within a web page with \( n \) elements. The RTDM must be computed for all subtrees, and the running time becomes \( O(n^4) \). This paper proposes a new algorithm which computes the distance between all the subtrees in a tree in time \( O(n^2) \), which enables us to obtain better quality as well as better performance, on a DOM mining task. In addition, we propose a new tree edit-distance — SiSTeR (Similar Sibling Trees aware RTDM). This variant of RTDM allows considering the case were repetitious (very similar) subtrees of different quantity appear in two trees which are supposed to be considered as similar.

1 INTRODUCTION

There is a growing interest in discovering knowledge from complex data, which is organized as trees, rather than as a single relational table. This research is motivated by applications that manipulate molecular data, XML data and Web content. We are particularly motivated by modern web applications. The content of these applications is, invariably, automatically generated using templates, whose content is filled from databases, or web toolkits, such as Google Web Toolkit. Such HTML documents are incredibly complex. For example, the Google search page, which presents a simple form, which a user perceives as a few interface objects, contains about 100 objects and its maximal depth is 12. While automatically generated content tends to be complex, it also tends to be consistent. Thus, the same functional components will tend to have similar DOM (Document Object Model) structure. A key aspect of of understanding DOM structures is, therefore, finding repeating DOM structures. To find such structures, we propose a new algorithm for finding frequent trees, which are similar, but not necessarily identical.

Frequent tree mining algorithms search for repeating subtree structures in an input collection of trees. These algorithms vary in the restrictions that the repeating structure must adhere to, and in the type of trees that are searched. These types include bottom-up subtrees in ordered, labeled trees (Luccio et al., 2001), induced subtrees, (Abe et al., 2002; Zaki, 2002), unordered trees (Luccio et al., 2004; Zaki, 2004) and embedded subtrees (Zaki, 2004). A good overview of frequent tree mining may be found in (Chi et al., 2005). For DOM structure mining, we are interested in a particular tree mining scenario. The trees are rooted, labeled and ordered. Unlike the algorithm in (Luccio et al., 2001) the patterns we seek are similar, but not identical.

There is also extensive prior research on similarity between trees and on tree edit distance algorithms. The prevalent definition of edit distance for labelled ordered trees was proposed by (Tai, 1979). For unordered trees the problem is known to be NP-hard (Bille, 2005). For ordered trees, on the other hand, polynomial algorithms exist (Tai, 1979; Zhang and Shasha, 1989). Several researchers have identified restrictions to this definition of edit distance. One example is the constrained edit distance, that was studied for ordered trees in (Zhang, 1995) and for unordered trees (Zhang, 1996). In (Lu, 1984), a distance metric based on node splitting and merging is defined.
A better notion of tree distance for mining the web is the top-down edit distance (Selkow, 1977), in which insertions and deletions are restricted to the leaves of the trees. A variant of this definition, the restricted top-down distance (Reis et al., 2004), is even more suitable for web mining, because it captures the process of building web pages.

The setting of DOM mining prescribes the type of trees we are working with. The repeating subtrees should include the actual content of the Web page. The internal nodes are often a collection of DIV and SPAN elements that can be aligned fortuitously. Thus, the subtrees are bottom-up, in principle, but small differences between trees are acceptable. The acceptable differences, or edit operations, are also restricted. The prevalent notion of edit distance does not match our intuition about the differences between HTML structures. For example, consider a complex control embedded in a container element. The edit distance between the control and its container is very small. That is, it is quite difficult to isolate the control from its container. This distinction is enabled by the Restricted Top-Down edit Metric (RTDM) (Reis et al., 2004), because of the restrictions it places on the permitted edit operations.

Given a collection of trees, we aim to find all the repeating subtrees. A naive algorithm for finding such repeating structure may be

1. For each pair of nodes in all input trees, compute the RTDM of the subtrees rooted at these nodes
2. Cluster subtrees based on the computed distances
3. Output significant clusters

Unfortunately, step (1) is computationally expensive. Given two trees with input size $n_1$ and $n_2$, the running time is $O(n_1^2 \cdot n_2^2)$, i.e., squared in the size of the input trees. Note that if we look for repeating structures on a single tree with $n$ nodes, e.g., one web page, the running time is $O(n^3)$.

Our first main contribution is a new algorithm that given two trees, computes the RTDM distances between all the subtrees in the first tree and all the subtrees in the second tree in time $O(n_1 \cdot n_2)$. To find the repeating subtrees in a single input tree with this algorithm would take time $O(n^3)$, rather than $O(n^3)$.

We further present a variant of RTDM. We consider the case where we want to be less sensitive to the number of similar sibling subtrees within two trees we compare. We present SiSTeR (Similar Sibling Trees aware RTDM), which is a variant of RTDM. This variant allows two trees to be considered as similar to each other, even when they differ with regards to the number of similar sibling subtrees within them (see a schematic example in Figure 1). For example, consider a citation of an article site. Each entry has the “Cited by” section. This subtree will have different number of child-subtrees (the cites) for each article. Regardless of their number, we would like to identify two “Cited by” subtrees as similar.

### 1.1 Organization

We formally define the restricted edit distance measure in section 2. The dynamic programming algorithm is presented in Section 3.1. We embed this algorithm in the context of a frequent tree mining algorithm in 3.3. In Section 3.2 we present and discuss our new RTDM variant, SiSTeR. We discuss two DOM mining applications, DOM structure mining and DOM pattern search, in Section 4. Section 5 provides experimental results, assessing the quality of our algorithm and the SiSTeR metric.

## 2 PRELIMINARIES

This section lays the formal infrastructure for our discussion. We consider rooted-ordered-labelled trees.

Within our framework some manipulations are allowed on trees. The allowed edit operations are somewhat different than that of standard operations and best suit our setting. The operations allowed in our framework are delete, insert and replace for subtrees. For two trees $T_1 = (V_1, E_1, L)$ and $T_2 = (V_2, E_2, L)$ and two vertices $v_1 \in V_1$ and $v_2 \in V_2$ we define the replace operation by $T_1(v_1) \rightarrow T_2(v_2)$ to be the tree $T_1$, when taking out the subtree $T_1(v_1)$ and replacing it with the subtree $T_2(v_2)$, where the order of $v_2$ as a child is the same order that $v_1$ had and the labels given by $L$ remain. When the context is clear we will write $T_1(v_1) \rightarrow T_2(v_2)$, for short. The delete operation, then, is defined to be $T_1(v_1) \rightarrow \lambda$ and the insert operation is defined to be $\lambda \rightarrow T_2(v_2)$, where $\lambda$ denotes the empty tree. See Figure 2 for an illustration of these edit operations.

Similar to other edit schemes, here too, we define a sequence of edit operations $S = s_1, \ldots, s_k$. The $S$-derivation of $T_1$ is defined to be the sequence of trees

![Figure 1: In some settings one would like to consider these two trees as very similar as they differ only with regards to the number of similar sibling subtrees within them.](image-url)
accepted by $\mathcal{T}_1(s_1)(s_2) \ldots (s_k)$. If the resulting tree is $\mathcal{T}_2$ we say that $S$ is a derivation from $\mathcal{T}_1$ to $\mathcal{T}_2$ and we denote it by $\mathcal{T}_1 \xrightarrow{S} \mathcal{T}_2$.

We define a cost function $\gamma$, which assigns a real number to each edit operation. This cost function is constrained in our framework to be a distance metric. The cost for a sequence $S$ is simply defined to be $\gamma(S) = \sum_{v \in V} \gamma(v)$. We then define the edit distance between two trees $\mathcal{T}_1$ and $\mathcal{T}_2$ to be the lowest-cost $S$-derivation from $\mathcal{T}_1$ to $\mathcal{T}_2$. That is:

$$D(\mathcal{T}_1, \mathcal{T}_2) = \min_{S: \mathcal{T}_1 \xrightarrow{S} \mathcal{T}_2} \{ \gamma(S) \}.$$  

(1)

In order to proceed, we broaden our definition to (directed-ordered-labelled) forests. A forest is a set of trees. The forests we are interested in are ordered forests, which means that the set of trees is ordered. All our definitions generalize naturally from trees to forests (including those of $S$-derivation, $\gamma$, and $D$, although the operations are still only defined for a single connected tree at a time). Given a tree $\mathcal{T} = (V, E, L)$, for any $v \in V$ denote $\mathcal{T}(v)$ to be the forest which consists of all the subtrees of $\mathcal{T}$ with the children of $v$ as their roots, with the order of the trees in the forest remains as the order of their roots as children of $v$.

Whereas prior top-down edit distance metrics are defined as operations on nodes, we define the edit distance in terms of operations on subtrees. Nonetheless, this definition differs from the top-down edit distance definition in (Selkow, 1977) only in the relabel operation, and it is identical to RTDM (Reis et al., 2004).

### 3 Scientific Contribution

#### 3.1 An All-Subtree Edit Distance Algorithm

The following is straightforward:

**Lemma 1.** For any two non-empty trees $\mathcal{T}_1 = (V_1, E_1, L)$ and $\mathcal{T}_2 = (V_2, E_2, L)$ and two vertices $v_1 \in V_1$ and $v_2 \in V_2$ it holds that:

$$D(\mathcal{T}_1(v_1), \mathcal{T}_2(v_2)) = \begin{cases} \gamma(\mathcal{T}_1(v_1) \rightarrow \mathcal{T}_2(v_2)) & L(v_1) \neq L(v_2) \\ D(\mathcal{T}_1(v_1), \mathcal{T}_2(v_2)) & \text{otherwise} \end{cases}$$  

(2)

where, the distance between two forests is defined as follows. For $h \in \{1, 2\}$ let $\mathcal{F}_h$ be a forest whose roots are $v_h^1, v_h^2, \ldots, v_h^n$, and for each $h$ denote by $\mathcal{F}_h^{i\rightarrow}$ the forest whose roots are $v_h^i, \ldots, v_h^n$, then:

$$D(\mathcal{F}_1, \mathcal{F}_2 \rightarrow) = \sum_{i=1}^{n_{\mathcal{F}_1}} \gamma(\mathcal{F}_1(v_i^1) \rightarrow \mathcal{F}_2(v_i^2))$$

$$D(\mathcal{F}_1, \mathcal{F}_2) = \sum_{i=1}^{n_{\mathcal{F}_2}} \gamma(\mathcal{F}_1(v_i^1) \rightarrow \mathcal{F}_2(v_i^2)) + D(\mathcal{F}_1^{v_i^1 \rightarrow}, \mathcal{F}_2^{v_i^2 \rightarrow})$$

$$D(\mathcal{F}_1, \mathcal{F}_2) = \min \left\{ \begin{array}{l} \gamma(\mathcal{F}_1(v_i^1) \rightarrow \mathcal{F}_2(v_i^2)) + D(\mathcal{F}_1^{v_i^1 \rightarrow}, \mathcal{F}_2^{v_i^2 \rightarrow}) \\ \gamma(\mathcal{F}_1(v_i^1) \rightarrow \mathcal{F}_2(v_i^2)) + D(\mathcal{F}_1^{v_i^1 \rightarrow}, \mathcal{F}_2^{v_i^2 \rightarrow}) \end{array} \right\}$$

(3)

To compute the edit distance of every pair of subtrees in two input trees efficiently, we adopt a dynamic programming approach. Prior algorithms (Selkow, 1977; Reis et al., 2004) begin at the root of the tree and follow the structure of the tree down. Our algorithm, on the other hand, uses a bottom-up approach. The challenge in the bottom up approach is that we do not know which subtrees to match. We, therefore, must match all subtrees to each other, which forms the basis of the all-subtree computation. As the computation moves up the tree, the constraints due to tree structure are enforced.

To compute the edit distance we consider, for any vertex in the tree $\mathcal{T}$, the subtree rooted at this vertex as a reversed pre-order sequence of vertices. (Note that this is not equal to post-order as the right-most child will appear first in our case.) Let $v_i$ be the $i$th vertex in that order ($i \geq 1$).

Given two trees $\mathcal{T}_1$ and $\mathcal{T}_2$, for any $h \in \{1, 2\}$ denote the index of the first child by

$$C_h(i) = \begin{cases} i-1 & \text{if } v_i \text{ of } \mathcal{T}_h \text{ has children} \\ 0 & \text{otherwise} \end{cases}$$

(4)

and the index of the previous sibling by

$$I_h(i) = i - |\mathcal{T}_h(v_i)|$$

(5)

We then define the two following matrices of dimensions $(n_1+1) \times (n_2+1)$, where the first, $M_V$, is aimed to capture the required distances between each two subtrees, and the second, $M_T$, assists the computation of the first by holding the distance between the forests that precedes the relevant trees. We define $M_V(0,0) = M_T(0,0) = 0$, and:

$$\begin{array}{l}
\end{array}$$

240
We present a variant of RTDM that makes our measure even more compatible with DOM applications. Our Similar Sibling-Trees Aware RTDM (SiSteR) is a variant in which multiple subtrees are treated as a set regardless of their number. In many websites sibling subtrees might be very similar, and do not impact similarity to other trees. Forum threads are a good example. In forums, the number of posts in a thread should not influence the similarity to other threads.

SiSteR presents two additional operations to the standard edit operations: one-to-many-replace and many-to-one-replace. The semantics of these operations is to allow a series of consecutive replaces of one subtree with many subtrees (rather than replace and then a row of inserts or deletes in standard RTDM). For these operations, the cost is defined to be the sum of the many replaces occurred. Note that the replace operation is a special case of many-to-one-replace and one-to-many-replace. Thinking of strings, this allows distance 0 between the string a and the string aaaaaa, unlike the standard edit-distance which requires 4 insert-operations. Here a one-to-many-replace operation with cost 0 for each of the replace operation allows “similar-sibling awareness”. A page in an article citation site on some paper will have different number of “Cited by” entries. Still, the subtree representing this “Cited by” part should be considered as similar to the same parts in different pages where this number might be completely different. We denote the SiSteR edit distance by $D'$.

We revise our all-subtree distance algorithm to use SiSteR. In addition to $M_F$ and $M_V$ we will also calculate $M_{OTM}$ and $M_{MTO}$ in the following way:

$$i, j > 0 \quad M_{OTM}(i, j) = M_V(i, j) + \min \begin{cases} M_F(i_1(i), i_2(j)) \\ M_{OTM}(i_2(j)) \end{cases}$$ (7)

$$i, j > 0 \quad M_{MTO}(i, j) = M_V(i, j) + \min \begin{cases} M_F(i_1(i), i_2(j)) \\ M_{MTO}(i_1(i), j) \end{cases}$$ (8)

In Equation 6 we insert the following change:

$$(8) \quad i, j > 0 \quad M_{VT}(i, j) = M_V(i, j) + \min \begin{cases} M_F(i_1(i), i_2(j)) \\ M_{MTO}(i_1(i), j) \\ M_{OTM}(i_2(j)) \end{cases}$$ (9)

We have the following version of Lemma 2

Lemma 3. Given two directed-ordered-labelled trees $T_1 = (V_1, E_1, L)$ and $T_2 = (V_2, E_2, L)$, the algorithm above computes $D'(T_1(v), T_2(u))$ for every $v \in V_1$ and $u \in V_2$ (and in particular $D'(T_1, T_2)$) correctly with computation complexity $O(n_1 \cdot n_2)$. 

3.3 A Frequent Tree Algorithm for Similar Occurrences

We propose an algorithm for finding sets of subtrees, such that each set contains a number of subtrees which are similar to each other. Thus, the required output is a meaningful clustering of bottom-up subtrees, in which the similarity measure is the RTDM. Given the All-Subtree Edit Distance Algorithm presented in Section 3.1 the frequent tree algorithm is straightforward.

Given the input tree perform the following:

1. Run the All-Subtree Edit Distance Algorithm as appears in Section 3.1 getting the distance matrix between every two subtrees in the input tree.
2. Based on this matrix, cluster the subtrees. We use a clustering approach from (Koontz et al., 1976).
3. Using thresholds, output the significant clusters.
4 APPLICATIONS

We discuss two applications for DOM mining.

4.1 DOM Structure Mining

The ability to efficiently find repeating structure in trees has immediate applications for mining Web applications. Several classes of constructs common to Web applications manifest as repeating DOM structures, including controls (e.g., the video controls common to YouTube.com site), records (e.g., search results in Google, items for purchase in Amazon and videos in YouTube) and containers (For example, in YouTube the Videos Being Watched Now and Most Popular are containers.)

The algorithm was activated on the YouTube.com site. Our experiment demonstrates that the algorithm can be used to find all three types of structures. Controls are easy to find, since the distance between the entire cluster of subtrees is 0. Records can be found by allowing clusters with some dissimilarity, we typically use 20% of the combined length of the subtrees as a distance threshold. Containers are the most difficult to identify. Nevertheless, our algorithm can often find containers using a higher distance threshold, e.g., 60% of the subtree sizes. Another approach for finding containers is to use the headers, which are more similar, and identify the container from the header. In YouTube, for which we used a threshold of 20% one gets the container’s headers to clustered together. The best method might combine information from headers and complete subtrees.

4.2 DOM Structure Search

The All-Subtree Edit Distance algorithm is further applicable to the problem of searching the web for a predefined DOM structure. In this use case, the user, or an application, seeks a known DOM structure, i.e., a pattern, in a collection of web pages. However, the pattern may be inexact. Applications that benefit from efficient search for inexact patterns include mashups, article extraction, and web automation.

It is straightforward to return the desired search results. The time for this algorithm is $O(n \cdot k)$, where $n$ is the size of the page and $k$ is the size of the pattern. Assuming that the size of the pattern is small and independent of $n$, the algorithm is linear in the size of the input tree.

5 RESULTS

5.1 Performance of the All-subtree Edit Distance Algorithm

In this section, we assess the computation time required to compute a complete distance matrix for DOM structures on a single web page. We compare the proposed all-subtree edit distance algorithm from Section 3.1, with applying the top-down algorithm (RTDM) (Reis et al., 2004) for all subtrees. The results are presented in Table 1. Times shown are in seconds. With the exception of the Amazon page, we used the home page of the application. On Amazon, the page we used is the results of searching for the keyword “spectrum”.

<table>
<thead>
<tr>
<th>Site</th>
<th>Size</th>
<th>RTDM</th>
<th>All-Subtree</th>
</tr>
</thead>
<tbody>
<tr>
<td>YouTube.com</td>
<td>1809</td>
<td>9.2495856</td>
<td>0.781215</td>
</tr>
<tr>
<td>Marriott.com</td>
<td>2180</td>
<td>12.2494512</td>
<td>1.4530599</td>
</tr>
<tr>
<td>Amazon.com</td>
<td>3848</td>
<td>45.3417186</td>
<td>3.4842189</td>
</tr>
<tr>
<td>Google.com</td>
<td>669</td>
<td>0.7968393</td>
<td>0.1093701</td>
</tr>
<tr>
<td>iGoogle.com</td>
<td>1000</td>
<td>2.7655011</td>
<td>0.2187402</td>
</tr>
</tbody>
</table>

5.2 Results for DOM Record Mining

This section evaluates our system as a tool for DOM record mining. In our test we chose pages that contained both lists and tables of items, mainly from previous works (Liu et al., 2003; Park and Barbosa, 2007; Zhai and Liu, 2005). We compared the precision and recall of our algorithm to DEPTA (Zhai and Liu, 2005). We refer the reader to that work for comparison to other alternatives. Table 2 shows a summary of the set of pages in our experiment.

<table>
<thead>
<tr>
<th>Cnt</th>
<th>Corr Frequent</th>
<th>Corr DEPTA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Frequent</td>
<td>DEPTA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>99.5%</td>
</tr>
<tr>
<td>200</td>
<td>200</td>
<td>100%</td>
</tr>
<tr>
<td>300</td>
<td>300</td>
<td>100%</td>
</tr>
</tbody>
</table>

With recall of 99% and precision of 100%, our performance is superior to that of (Park and Barbosa, 2007), which reported 85.7% recall and 100% precision. The DEPTA system failed to run on these examples.
In general, one can say our results improve upon prior systems in all respects. The main advantages of our results are better recall in the harder cases, and no over segmentation of the different sets.

Table 2: Experimental Results for Record Retrieving.

<table>
<thead>
<tr>
<th>URL</th>
<th>Tree Size</th>
<th>Cmt</th>
<th>Frequent Similar Trees</th>
<th>DEPTA</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.amazon.com">www.amazon.com</a></td>
<td>3359</td>
<td>16</td>
<td>3.67</td>
<td>0</td>
</tr>
<tr>
<td>forums.geonoo.com</td>
<td>2837</td>
<td>25</td>
<td>3.28</td>
<td>18(4)</td>
</tr>
<tr>
<td>forums.sun.com</td>
<td>1729</td>
<td>15</td>
<td>0.97</td>
<td>15(3)</td>
</tr>
<tr>
<td>shoutwire.com</td>
<td>3543</td>
<td>20</td>
<td>4.6</td>
<td>20</td>
</tr>
<tr>
<td>messages.yahoo.com</td>
<td>2017</td>
<td>38</td>
<td>1.25</td>
<td>37</td>
</tr>
<tr>
<td><a href="http://www.gateway.com">www.gateway.com</a></td>
<td>1461</td>
<td>6</td>
<td>1.14</td>
<td>0</td>
</tr>
<tr>
<td>shop.ebay.com</td>
<td>3604</td>
<td>50</td>
<td>4.53</td>
<td>40</td>
</tr>
<tr>
<td><a href="http://www.google.com">www.google.com</a></td>
<td>848</td>
<td>11</td>
<td>0.19</td>
<td>0</td>
</tr>
<tr>
<td><a href="http://www.abt.com">www.abt.com</a></td>
<td>5408</td>
<td>40</td>
<td>10.62</td>
<td>40(9)</td>
</tr>
<tr>
<td><a href="http://www.allbbris.com">www.allbbris.com</a></td>
<td>3225</td>
<td>25</td>
<td>3.73</td>
<td>25</td>
</tr>
<tr>
<td>bobbsiscountermarine</td>
<td>1318</td>
<td>16</td>
<td>1.69</td>
<td>0</td>
</tr>
<tr>
<td><a href="http://www.compusa.com">www.compusa.com</a></td>
<td>2884</td>
<td>18</td>
<td>2.99</td>
<td>18(5)</td>
</tr>
<tr>
<td><a href="http://www.cooking.com">www.cooking.com</a></td>
<td>2130</td>
<td>23</td>
<td>1.67</td>
<td>23</td>
</tr>
<tr>
<td><a href="http://www.dealltime.com">www.dealltime.com</a></td>
<td>3588</td>
<td>13</td>
<td>0.53</td>
<td>14(4)</td>
</tr>
<tr>
<td><a href="http://www.drugstore.com">www.drugstore.com</a></td>
<td>1572</td>
<td>42</td>
<td>0.67</td>
<td>42(4)</td>
</tr>
<tr>
<td>magazinesofamerica</td>
<td>759</td>
<td>6</td>
<td>2.2</td>
<td>6(2)</td>
</tr>
<tr>
<td><a href="http://www.nextag.com">www.nextag.com</a></td>
<td>5351</td>
<td>30</td>
<td>8.72</td>
<td>30</td>
</tr>
<tr>
<td>nothingbutsoftware</td>
<td>3047</td>
<td>24</td>
<td>4.25</td>
<td>24(6)</td>
</tr>
<tr>
<td><a href="http://www.refurbdepol">www.refurbdepol</a></td>
<td>2890</td>
<td>15</td>
<td>4.81</td>
<td>10(5)</td>
</tr>
<tr>
<td>rochesterclothingc</td>
<td>1820</td>
<td>16</td>
<td>0.98</td>
<td>16(4)</td>
</tr>
<tr>
<td><a href="http://www.smartbargains.c">www.smartbargains.c</a></td>
<td>3095</td>
<td>24</td>
<td>3.06</td>
<td>0</td>
</tr>
<tr>
<td><a href="http://www.tigerdirect.c">www.tigerdirect.c</a></td>
<td>1527</td>
<td>20</td>
<td>0.93</td>
<td>20(5)</td>
</tr>
<tr>
<td>Sum/Average</td>
<td>2534</td>
<td>516</td>
<td>3.08</td>
<td>420</td>
</tr>
<tr>
<td>Recall</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Only for successful</td>
<td></td>
<td>99%</td>
<td>81%</td>
<td></td>
</tr>
</tbody>
</table>

5.3 Results for SiSTeR to Allow Detect Similarity of Forums

This section evaluate the SiSTeR variant of RTDM to allow to be aware to similar sibling subtrees when computing the similarity between DOM trees. In particular, when looking on forums’ DOM trees the amount of lines or quotes in different posts create a big difference between different posts. Here, the posts each of which might have very different number of lines or different number of quotes of other posts, should be discovered as similar. We have tested our algorithm using SiSTeR in comparison with the same algorithm using RTDM. We have checked the similarity between different posts in all of the following forums: forums13.itrc.hp.com, forums.oracle.com, hackquest.com, fdt.powerflasher.com/forum, and forum.projecteuler.net. In all these examples we noticed a significant reduction in the number of clusters that were discovered by using our similarity measure as a distance metric. For example, in the forum projecteuler.net the RTDM-based algorithm outputed 10 different clusters of posts and another 20% of posts that were in no cluster, while the SiSTeR-based algorithm reduced it to 4 clusters and 10% un-clustered posts.

REFERENCES


