2013 17th European Conference on Software Maintenance and Reengineering

# Feature Detection in Ajax-enabled Web Applications

Natalia Negara
*Department of Computing Science*
*University of Alberta*
*Edmonton, Canada*
*negara@ualberta.ca*

Nikolaos Tsantalis
*Department of Computer Science*
*and Software Engineering*
*Concordia University*
*Montreal, Quebec*
*nikolaos.tsantalis@concordia.ca*

Eleni Stroulia
*Department of Computing Science*
*University of Alberta*
*Edmonton, Canada*
*stroulia@ualberta.ca*

***Abstract*—In this paper we propose a method for reverse engineering the features of Ajax-enabled web applications. The method first collects instances of the DOM trees underlying the application web pages, using a state-of-the-art crawling framework. Then, it clusters these instances into groups, corresponding to distinct features of the application. The contribution of this paper lies in the novel DOM-tree similarity metric of the clustering step, which makes a distinction between simple and composite structural changes. We have evaluated our method on three real web applications. In all three cases, the proposed distance metric leads to a number of clusters that is closer to the actual number of features and classifies web page instances into these feature-specific clusters more accurately than other traditional distance metrics. We therefore conclude that it is a reliable distance metric for reverse engineering the features of Ajax-enabled web applications.**

***Keywords*-web page similarity metrics; hierarchical agglomerative clustering; L method; Silhouette coefficient.**

## I. Introduction

A feature represents a functionality that is defined by requirements and is accessible to users [1]. Extracting the features of a web application can be applied to the automatic generation and update of meta-keywords (used by search engines) and site maps; tasks which require significant manual effort, especially for web applications whose content and layout changes frequently due to their dynamic nature. A standard reverse-engineering approach [2], [3] involves systematically exercising the web application to collect the pages it serves, and clustering these pages in groups, each group representing a feature of the web application, available to its users through the pages contained in the group. The behavioral model inferred through this process represents the web application features (i.e., services) as groups of similar pages, and the control and data dependencies between them as the user interactions that lead from a page in one group to another (i.e, navigation model).

The widespread adoption of Web 2.0 technologies during the last decade has fuelled the development of web applications with dynamic content [4]. Exercising such applications to collect the pages they serve is challenging, since the number of pages that can be potentially produced is practically unlimited. This problem is now mitigated by the development of tools for automatically crawling and testing modern (Ajax) web applications, such as Crawljax [5]. However, the problem of feature extraction for such applications is still challenging, because of the complexity of the generated pages, which may include script and style segments, in addition to their HTML content. This complexity can make the task of page-similarity assessment, which is essential for page clustering, very difficult.

In the past, string-based edit distance metrics have been used with some success [6]–[10] to assess the similarity of web pages by treating the pages as sequences of HTML tags. In our approach, however, we adopt a tree-differencing algorithm that takes as input two DOM-tree instances and returns an edit script, which, if applied to the first tree can transform it into the second one. In the distance computation between DOM trees, the edit operations nested under the same path to the root are considered as a single composite edit operation, of equal weight to that of an individual edit operation.

As an example, imagine a scenario where a user visits a search page, provides the search criteria through some input widgets and clicks on the "search" button. In response to the user action, a new DOM tree is generated that contains a table component with the results of the query. The table component may provide dynamic features, such as data sorting, mouse hover and pop-up menus, which usually increase the number of generated DOM nodes. The search-results page is part of the general search feature offered by the web application. However, standard tree-differencing approaches would consider each additional DOM node as a distinct change and would likely produce a low similarity value between the original and the new DOM tree. The proposed similarity metric would summarize all additional nodes as part of a single composite change, thus likely considering the two pages as similar enough to be part of the same cluster.

To evaluate our approach, we compared the proposed metric with the Levenshtein distance, which has been widely used in the literature for computing the structural similarity of web pages treated as sequences of HTML tags, and a simplified tree-edit-operation based distance metric that does

1534-5351/13 $26.00 © 2013 IEEE
DOI 10.1109/CSMR.2013.25

not take into account composite changes. The metrics have been evaluated on sets of DOM states collected from three Ajax-enabled web applications, namely Garage.ca, Google Maps and Kayak.com, using the Crawljax [5] crawling engine. The images depicting the rendered DOM states (i.e, screenshots of the web pages rendered in the browser) have been manually clustered based on the feature they correspond to, as perceived by the authors of the paper. Using the resulting partitions of clusters as a reference, we compared the metrics with respect to their ability to produce the correct number of features using two different methods, namely the L method [11] and the Silhouette coefficient [12]. These methods can be used to automatically determine the number of clusters to be returned from the application of the hierarchical clustering algorithm on a given set of data points.

The rest of the paper is organized as follows: Section II presents an overview of the related work. In Section III we present our approach for detecting features. Section IV evaluates the proposed distance metric. Finally, we conclude in Section V.

## II. Related Work

Several reverse-engineering techniques have been proposed in order to support comprehension, maintenance and evolution of web applications. Ricca and Tonella [6] proposed a semi-automatic approach to identify static web pages with a similar structure that can be migrated to dynamic web pages. To identify the groups of web pages with a common structure, they applied an agglomerative clustering algorithm with the Levenshtein edit distance as the similarity metric between web pages. The clusters of similar pages are then selected by cutting the resulting dendrogram at a chosen level. Next, for all pages in the same cluster, a common candidate template is extracted to be used in the dynamic generation of web pages, along with variable information produced by comparing the original pages with that template.

Mesbah and van Deursen [13] used a schema-based clustering technique to categorize web pages with similar structures. The authors extract the navigational model of a web application and compute the Levenshtein edit distance between the reverse-engineered schemas of the web pages. The navigational path and schema-based similarities are then given as input to a clustering algorithm. The formed clusters are analyzed to find candidate user interface components to be migrated from a multi-paged web application to a single-paged Ajax interface.

A similar approach was suggested by De Lucia et al. [7] for the generalization of cloned patterns in reengineering processes. The Levenshtein distance between two web pages is separately computed at the structural (using string representations of the HTML tags of web pages) and content levels. For dynamic pages the similarity degree is also computed at the scripting code level. Two pages are considered as clones if their edit distance is lower than a given threshold.

Di Lucca et al. [8] suggested an approach to detect clone pages that have the same structure, but differ only in their content. In the proposed approach the sequence of tags of HTML and ASP pages are encoded into a string representation and the Levenshtein edit distance is computed between them. The authors defined a separate alphabet for both client and server pages, and each HTML or ASP tag is replaced with a corresponding alphabet character. This is done in order to take into account the different techniques and languages used to implement control components in these pages. Along with the Levenshtein based technique, the authors suggested detecting duplicate pages using a metric based on the frequency of each HTML tag inside a page. The evaluation results show that both methods are comparable, but have different computational costs.

To support the comprehension of legacy web applications, De Lucia et al. [9] proposed an approach that groups similar pages at structural level using the Winner-Takes-All clustering algorithm. The distance between both dynamic and static pages is computed using the Levenshtein edit distance, where the web pages are represented as string sequences of HTML tags. Like [8], the HTML tags of the pages are encoded into the symbols of an alphabet, which makes the computation more precise and faster.

Later, De Lucia et al. [10] suggested to group web pages that are similar at content level. The dissimilarity between web pages at content level is computed by combining the Levenshtein and the cosine distance between the vectors of the page in the latent structure of content [10]. The approach uses a weighted mean to combine these metrics. Similar pages are grouped by iteratively applying a graph-theoretic clustering algorithm, which takes as input a graph with web pages as nodes and edge weights corresponding to the combined dissimilarities between pairs of pages, and then constructs a minimal spanning tree (MST). Clusters are identified by pruning the edges of the MST that have a weight higher than a given threshold.

Di Lucca et al. [14] proposed a technique to decompose web applications into groups of functionally related components which employs an agglomerative hierarchical clustering algorithm. The coupling measure between interconnected components, which also considers the topology of connections, is used as a similarity metric. The output of the clustering algorithm represents a hierarchy of clusters, which is used to understand the structure of a web application. The cutting threshold for the produced dendrogram is selected based on a quality metric computed as the difference between intra- and inter-connectivity of the clusters.

The majority of the aforementioned works which are focused on grouping similar web pages employ the Levenshtein edit distance [15] as a similarity metric between

pages. The Levenshtein edit distance between two sequences is defined as the minimum number of insert, delete, and replace operations required to transform the first sequence into the second. The major limitation of this approach is that the web pages are converted into a sequence of HTML tags in order to be compared and thus nesting information cannot be utilized. Earlier work in our group [16] demonstrated the effectiveness of tree-differencing for recognizing similarities and differences between web pages in the context of page evolution; in fact the algorithm deployed in this study is a precursor of the algorithm we have adopted in this work.

## III. Feature Extraction

Our process for extracting features from a set of DOM trees consists of three steps. First, the distance between all pairs of DOM trees is calculated. Next, the computed distances are given as input to a hierarchical clustering algorithm, which produces a dendrogram of cluster merges based on the similarity of the DOM trees. Finally, we apply clustering analysis methods to determine the number of clusters corresponding to the actual features. In our experiments we applied two different methods, namely the L method [11] and the Silhouette coefficient [12].

### A. The Distance Metric

For the computation of distance between two DOM trees we employ *VTracker* [16], [17], which is a generic XML-document comparison algorithm based on the Zhang-Shasha tree-edit distance algorithm [18]. *VTracker* considers the input XML documents as *labeled ordered trees* and produces as output a *tree-edit sequence*, i.e., a sequence of edit operations that can be applied to transform the first tree into the second. The actual page representation fed as input to *VTracker* is a "cleaned up" version of the original DOM tree. The clean-up process involves the following steps.

1) The page content is removed, by deleting the text being present between opening and closing tags.
2) Scripts and CSS styling rules are eliminated, by completely deleting <*SCRIPT*> and <*STYLE*> tags.
3) In particular applications, special attributes with script values are removed. For example, in Google Maps we found several <*DIV*> tags where attributes, such as `jsprops`, `jsdisplay`, `jstcache`, `jsattrs`, `href`, `onlick`, had scripts as values that were slightly changing between subsequent DOM trees and artificially decreased the similarity of the DOM trees.

The edit script reported by *VTracker* consists of the following primary edit operations at node level:

1) match (i.e., found to be exactly the same) across both trees,
2) change from one tree to the second,
3) move from one location in the first tree to a different location on the second one,
4) remove from the first tree, and
5) insert in the second tree.

The distance between two trees $T_1$ and $T_2$ is defined as:

$$d_S(T_1, T_2) = \frac{|\mathit{diff}(T_1,T_2)|}{|\mathit{diff}(T_1,T_2)|+|\mathit{match}(T_1,T_2)|} \quad (1)$$

where $\mathit{diff}(T_1, T_2)$ is the set of change, move, delete and insert operations in the tree-edit sequence of $T_1$ and $T_2$ and $\mathit{match}(T_1, T_2)$ is the set of match operations in the tree-edit sequence of $T_1$ and $T_2$. The distance metric defined in (1) ranges over the interval $[0, 1]$. It is minimized when $|\mathit{diff}(T_1, T_2)|$ is equal to zero and maximized when $|\mathit{match}(T_1, T_2)|$ is equal to zero.

We define a *composite edit operation* as a subset of primary operations in $\mathit{diff}(T_1, T_2)$ where the affected nodes are nested under the same path to the root. More specifically, two primary edit operations on nodes $A$, $B$ belong to the same composite operation, if the path of node $A$ to the root is part of the path of node $B$ to the root. As a result, a composite edit operation may consist of different types of primary edit operations (e.g., node additions along with node changes), as long as the involved nodes are nested under the same path to the root. Figure 1 illustrates an example of a composite edit operation (highlighted in yellow) that took place between two DOM states of the JPetStore web application. As it can be observed, the path of node $A$ to the root is part of the path of node $B$ (and every nested node within $A$) to the root. Consequently, the insertion operations corresponding to the nodes highlighted in yellow constitute a composite edit operation. The change operation on node $C$ (highlighted in green) is an individual edit operation, since the path of node $A$ to the root is not part of the path of node $C$ to the root.

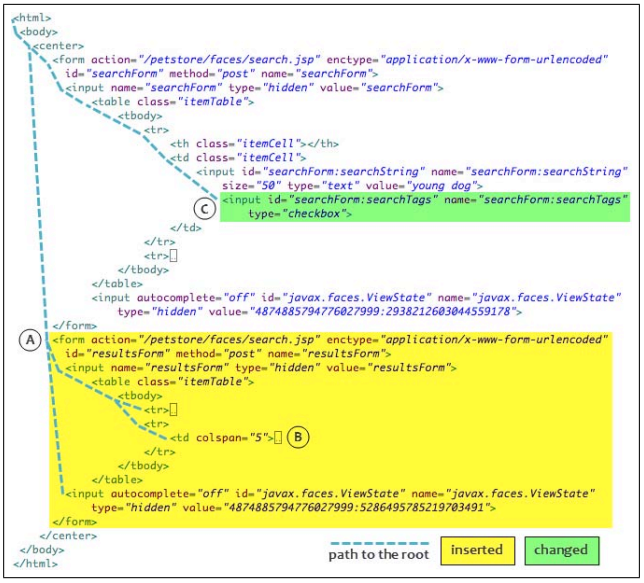



Figure 1: Example of a composite edit operation in JPetStore

By extracting the set of composite edit operations we can compute a new distance metric between two trees $T_1$ and $T_2$ defined as:

$$d_C(T_1,T_2) = \frac{|\textit{diff'}_{(T_1,T_2)}|+|\textit{comp}_{(T_1,T_2)}|}{|\textit{diff'}_{(T_1,T_2)}|+|\textit{comp}_{(T_1,T_2)}|+|\textit{match}_{(T_1,T_2)}|} \quad (2)$$

where:

- $\textit{comp}(T_1,T_2)$ is the set of composite edit operations extracted from the tree-edit sequence of $T_1$ and $T_2$;
- $\textit{diff'}(T_1,T_2) = \textit{diff}(T_1,T_2) \setminus \textit{editOps}(T_1,T_2)$;
- $\textit{editOps}(T_1,T_2) = \bigcup_{x \in comp(T_1,T_2)} x$ is the union of all edit operations in the set of composite edit operations.

In the distance metric defined in (2), the number of primary edit operations belonging to composite edit operations is actually replaced with the number of composite edit operations. In the example of Figure 1, the number of matched nodes is 98, the number of inserted nodes is 34 and there is only 1 changed node. Based on these numbers, $d_S$ is equal to $35/133 = 0.26$, while $d_C$ is equal to $2/100 = 0.02$, since $|\textit{diff}(T_1,T_2)| = 35$, $|\textit{editOps}(T_1,T_2)| = 34$, $|\textit{diff'}(T_1,T_2)| = 1$ and $|\textit{comp}(T_1,T_2)| = 1$. As a result, $d_C$ is more robust to composite edit operations by reducing their weight.

### B. The Clustering Algorithm

We adopted the hierarchical clustering algorithm to group DOM trees corresponding to the same feature, because it has a number of advantages when compared to partitioning or density-based clustering algorithms. Partitioning algorithms, like K-means [19], [20] for example, require as input the the number of clusters, and randomly distribute the data points into $k$ groups. Next, the algorithm examines the value of a fitness function (which usually corresponds to the cohesion of the clusters) and tries to optimize it by rearranging the data points in each iteration. The specification of $k$ is not intuitive in our problem, since we do not have *a priori* knowledge of the number of features existing in a web application. Some heuristic approaches attempt to automatically determine $k$ by comparing the quality of the clusters produced by the execution of K-means for different values of $k$ [21]. However, this may introduce a significant computational overhead if the number of DOM trees given as input is large (i.e., $k$ value should range from 2 to the number of input DOM trees). On the other hand, the hierarchical clustering algorithm has to be executed only once.

Additionally, because of the random initialization of clusters, K-means may reach a local optimum and will note always produce the same final result [22]. More generally, partitioning algorithms are not robust against noise (i.e., outliers) [23] and may return poor clustering results. Finally, K-means is typically applied on data points with specific coordinates in a feature space (e.g., Cartesian plane), while the input in our problem is given in the form of distances between pairs of DOM trees.

Density-based clustering algorithms, like DBSCAN [24], identify clusters as dense areas of data points [25]. The data points that satisfy a certain density criterion (i.e., minimum number of data points in a dense area) are grouped within a predefined distance threshold. Again, the manual specification of the density criterion is not intuitive in our problem, since there is no clear mapping between *density* and the number of DOM trees corresponding to a feature. Another limitation of this clustering method is the existence of *bridges* (i.e., data points which are equally close to two dense areas). This could potentially lead to the merging of clusters of DOM trees with different features.

The hierarchical agglomerative clustering algorithm takes as input the distances between each pair of data points (DOM trees in our case) and starts by assigning each entity to a single cluster. In each iteration of the algorithm the two closest clusters are merged. Finally, the algorithm terminates when all entities are placed in a single cluster, which forms the root of a hierarchy of clusters represented as a dendrogram. The actual clusters can be determined at the merging points. This algorithm is deterministic because it does not involve any random steps and finite because it does not contain an optimization process that could fall into local optima. Finally, it does not assume any input parameters.

The hierarchical clustering algorithm requires a *linkage criterion* that will guide the selection of the clusters to be merged. The most commonly used criteria are

- complete-linkage (the maximum distance between the data points of two clusters),
- single-linkage (the minimum distance between the data points of two clusters) and
- average-linkage (the average distance between the data points of two clusters).

The single-linkage approach tends to form highly separated clusters whereas complete-linkage tends to form more tightly centered clusters [26]. In the context of DOM-tree similarity, most of the pages of a web application have several elements in common (e.g., header, footer and menu), which implies that the distances between two DOM trees are relatively small (i.e., there do not exist highly distant clusters of pages). Therefore, an approach that tends to form more tightly centered clusters (i.e., complete-linkage) seems to be more appropriate, since this specific problem requires to group the most similar pages (tight clusters) among rather similar pages.

### C. Determining the Number of Clusters

The dendrogram resulting from the application of the hierarchical agglomerative clustering, requires a cut-off threshold value to determine the clusters. Previous approaches used either a predefined threshold [6] or required a user-defined

threshold [7], [8], [13]. In our approach, we apply clustering analysis methods to determine the number of clusters and thus deduce the cut-off threshold.

*1) The L method:* The L method [11] is an efficient technique for determining the correct number of clusters in hierarchical clustering and segmentation algorithms. The L method makes use of the same evaluation function that is used by the hierarchical algorithm during clustering (i.e., the merge distance) in order to construct an evaluation graph where the $x$-axis is the number of clusters and the $y$-axis is the value of the merge distance at $x$ clusters. The knee or the point of maximum curvature of this graph is used to estimate the number of correct clusters that should be returned as a solution to the clustering problem.

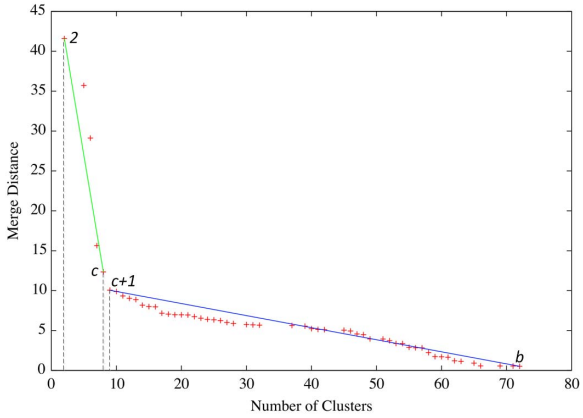



Figure 2: JPetStore evaluation graph

Figure 2 shows the evaluation graph produced by the hierarchical clustering algorithm using the complete-linkage criterion for the JPetStore web application. Three distinct regions can be identified on the graph: an inclined region of data points on the left of a graph, a flat region of data points that can be shaped as an almost straight line on the right of the graph, and a curved region between them. The nearly straight line on the right demonstrates that there are a lot of similar clusters that should be merged together. The sharply increasing region to the left (moving from the right) denotes that at this point dissimilar clusters are merged, thereby the quality of the clusters is being decreased. Hence, a reasonable number of clusters is in the curved area, or "knee" of the graph. Clusterings in the knee region contain a balance of clusters that are both highly homogeneous, and also dissimilar to each other.

The L method finds a pair of lines that most closely fit the data points. Each line must start at either end of the data points and must include at least two points. Both lines together cover all data points on the graph. The method does not take into account the data point that corresponds to the final cluster (when all clusters are merged into one), therefore the $x$-values range from 2 to $b$ (which is equal to the total number of elements given as input to the clustering algorithm), and the total number of data points on the graph is $b-1$. If we consider that the data points are partitioned at $x = c$ (see Figure 2), then $L_c$ and $R_c$ are the left and right sequences of data points, respectively. Left sequence $L_c$ includes data points $x = 2..c$ and right sequence $R_c$ comprises data points $x = c+1..b$. The total root mean squared error $RMSE_c$ is defined as:

$$RMSE_c = \frac{c-1}{b-1} \times RMSE(L_c) + \frac{b-c}{b-1} \times RMSE(R_c) \quad (3)$$

where $RMSE(L_c)$ is the root mean squared error of the best fit line for the left sequence of data points in $L_c$ and $RMSE(R_c)$ is the root mean squared error of the best fit line for the right sequence of data points in $R_c$ [11]. The weights are proportional to the lengths of $L_c$ $(c-1)$ and $R_c$ $(b-c)$, respectively. The L method seeks the value of $c$, $\hat{c}$, such that $RMSE_c$ is minimized. The location of the knee at $x = \hat{c}$ is used as the number of clusters to return.

In order to compute $\hat{c}$, the L method iterates over the values of $c = 3..b-2$ and forms a pair of lines (the first line by joining the points corresponding to $x$-values 2 and $c$ and the second one by joining the points corresponding to $x$-values $c+1$ and $b$) in order to compute the $RMSE_c$ value for each possible value of $c$.

*2) The Silhouette coefficient:* The Silhouette coefficient [12] is a clustering evaluation metric that describes how well each data point is located within a partition of clusters. It takes as input dissimilarities (i.e., distances) between pairs of data point and a partition of clusters produced by any clustering algorithm. For every data point $i$ in a set of data points $I$, it computes $a(i)$ as an average dissimilarity between $i$ and any other data point within the same cluster $A$. The value $a(i)$ demonstrates how well $i$ fits within its cluster (the smaller the value, the better the data point is fitted). Next, for every cluster $C$ which doesn't contain $i$, it computes the average dissimilarity of $i$ with all data points from $C$. The cluster with the lowest average dissimilarity is called the *nearest neighbour* and the average dissimilarity between $i$ and this cluster is denoted as $b(i)$.

The Silhouette $s(i)$ for data point $i$, and the average Silhouette coefficient $\bar{s_k}$ over all data points in $I$ in a partition of $k$ clusters are computed as:

$$s(i) = \frac{b(i)-a(i)}{max(a(i),b(i))} \quad (4) \qquad \bar{s_k} = \frac{1}{|I|} \sum_{i=1}^{|I|} s(i) \quad (5)$$

If cluster $A$ contains only one data point, $s(i)$ is set to be equal to zero. The value of $s(i)$ ranges over the interval $[-1, 1]$; the closer $s(i)$ is to one, the more appropriately the data point $i$ is clustered within the partition. The larger the value of $\bar{s_k}$, the better the quality of the clustering is.

## IV. Evaluation

The research question that we examine in this paper is whether clustering web application pages into features

improves when a composite-change-aware tree-edit distance metric ($d_C$) is used, as compared to either a tree-edit distance metric that treats all structural changes uniformly ($d_S$) or to the Levenshtein string-edit distance metric ($d_L$) applied to "flat" sequences of HTML tags.

### A. Experimental Setup

We have experimented with three different web applications: an online shopping application (garage.ca), a map-navigation and directions application (googlemaps.com) and an online travel booking application (kayak.com). All three web applications are implemented using Ajax, which allows to dynamically update their DOM trees at runtime.

The examined web applications have been analyzed with Crawljax [5], which is a tool for automatically crawling and testing Ajax-enabled web applications. Crawljax can crawl any Ajax-enabled web application by firing events and filling in form data. It creates a *state-flow* graph of the dynamic DOM states and the transitions between them. Crawljax was configured to collect 100 different states (at maximum) starting from the main page of the web applications. Moreover, in the case of forms requiring user input, we configured Crawljax to provide meaningful information instead of random strings.

For each collected DOM state we took a screenshot of the corresponding web page as rendered in the browser. The two first authors of the paper independently categorized the screenshots according to their visual perception of the feature that they offer. Next, the two authors merged their results by reaching a common consensus in the cases of a different feature interpretation. More specifically, in cases where one of the authors grouped sub-clusters of features into a single higher-level feature, while the other one created a separate group for each of these lower-level features, we decided to adopt the latter approach (i.e., a fine-grained decomposition of features). The merged results have been considered as the actual clusters of similar DOM trees (*reference*) based on which the accuracy of the examined distance metrics was evaluated. Table I shows the features extracted by the authors. Each cluster of DOM trees has been assigned with a name describing the feature provided by the web application (again as perceived by the two authors). The initial agreement between the two authors (before merging) based on the Jaccard Index[1] is equal to 0.887 for Garage, 0.890 for Google Maps and 0.993 for Kayak.

Table II lists the number of DOM states collected by Crawljax, along with the average, median and standard deviation of the number of nodes in DOM states and the depth of DOM structures for each of the examined web applications. From the examined web applications, Garage and especially Google Maps exhibit a low variation in the size as well as the depth of their DOM trees based on the corresponding standard deviation values. This means that the extracted DOM trees for these two web applications have a quite similar structure. On the other hand, Kayak exhibits the highest variation in both size and depth of its DOM trees. The reason behind this difference in the variation of size and depth lies in the nature of the examined web applications. Kayak is a search engine for travel offers and thus search queries may produce web pages with a very long list of results that vary significantly in size, depth and structural complexity in general. Google Maps, on the other hand, has an almost identical presentation structure (i.e., a query panel and a map for displaying the results of the queries) among the different features being offered. Finally, Garage.ca lies somewhere in the middle, since it is an online shop for clothes which has a relatively consistent presentation structure between web pages corresponding to the same feature.

Table I: Manually extracted features

| Web app. | Features |
|---|---|
| Google Maps | (1) Main search page, (2) Input origin and destination for directions, (3) Display directions by car, (4) Display directions by public transport |
| Garage | (1) Main page, (2) Gift cards and certificates, (3) Sign-in page, (4) Purchase gift card, (5) Locate store on Google Maps, (6) Online store FAQ, (7) Contact Garage, (8) Tops, (9) Coats, (10) Tanks, (11) Long sleeves tees, (12) Shirts, (13) Jeans, (14) Accessories, (15) Sleepwear, (16) Sale items, (17) Item detailed view, (18) Technical help, (19) Payment help |
| Kayak | (1) Search for flights, (2) Login page, (3) Password reminder, (4) Help, (5) Search for hotels, (6) Hotel search results, (7) Search for cars, (8) Car search results, (9) Car advanced search, (10) Search for deals, (11) Deals search results, (12) Flight search results, (13) Trip planner, (14) Find a Kayak booking, (15) "More" page, (16) Hotel advanced search |

To compare the structural similarity of two DOM trees using the Levenshtein distance we follow the same approach used in the literature [6]–[8]. We extract the sequence of HTML tags from the DOM trees by removing the content (i.e., text within tags) and the attributes inside the tags. On the contrary, when computing the tree-edit distance metrics ($d_S$ and $d_C$) the attributes inside the tags are not removed (with the exception of attributes having script code as values). By considering each tag as a token, the Levenshtein distance returns the minimum number of edit operations (i.e., insertion, deletion, and replacement of tokens) required to transform the first sequence of tags into the second. To normalize the computed distance within the $[0, 1]$ range, it should be divided by the maximum attainable value of edit operations, which is equal to the length of the largest input sequence. As a result, the normalized distance metric for two sequences of tags $S_1$ and $S_2$ is defined as:

[1]For two clusterings $C$ and $C'$ the Jaccard Index is defined as $J(C, C') = \frac{N_{11}}{N_{11}+N_{10}+N_{01}}$, where $N_{11}$ the number of point pairs in the same cluster under both $C$ and $C'$, $N_{10}$ the number of point pairs in the same cluster under $C$ but not under $C'$, and $N_{01}$ the number of point pairs in the same cluster under $C'$ but not under $C$.

Table II: Number of states, DOM size and depth for the examined web applications.

| Web application | # of states | Average # of nodes | Median (nodes) | Standard deviation (nodes) | Average DOM depth | Median (depth) | Standard deviation (depth) |
|---|---|---|---|---|---|---|---|
| Google Maps | 45 | 968 | 1080 | 231.83 | 20 | 20 | 0.59 |
| Garage.ca | 66 | 547 | 540 | 111.37 | 12 | 12 | 1.60 |
| Kayak | 78 | 894 | 887.5 | 541.65 | 26 | 29 | 6.36 |

$$d_L(S_1, S_2) = \frac{Ld(S_1,S_2)}{max(length(S_1),length(S_2))} \quad (6)$$

where $Ld(S_1, S_2)$ is the Levenshtein distance between sequences $S_1$ and $S_2$ and $length(S)$ is the number of tags in sequence $S$.

### B. Experiment results

For each distance metric $d_S$, $d_C$, and $d_L$, we have applied the hierarchical agglomerative clustering algorithm (using the complete-linkage criterion) on the set of DOM trees collected by Crawljax for the examined web applications.

Using the partitions of clusters and merge distances obtained from the resulting dendrogram for each examined web application and distance metric:

1) We applied the L method [11] to the evaluation graphs (Figure 3) generated using the distances produced by $d_S$, $d_C$, and $d_L$ measures as input to the hierarchical clustering algorithm. For each evaluation graph, we computed the value of $x = \hat{c}$ that minimizes the $RMSE_c$ value and constitutes a reasonable number of clusters to be returned according to the L method.
2) For each partition $P_k$ with $k$ number of clusters, where $k$ ranges from 2 to $n-1$ ($n$ is the total number of extracted DOM trees for a web application) we computed $\bar{s_k}$. A reasonable number of clusters $k_\alpha$ to be returned corresponds to the maximum value $\alpha$ over all Silhouette coefficients $\bar{s_k}$:
$\alpha = \max(\{\bar{s_k} : k = 2, ..., n-1\})$ (7)

The predicted number of clusters returned by both methods for each distance metric and web application were used to produce partitions with $\hat{c}$ and $k_\alpha$ number of clusters, respectively. To evaluate the quality of the resulting partitions we computed precision and recall for each partition. In order to compute precision and recall we have to provide a definition of *True Positives*, *False Positives* and *False Negatives*. Let *reference* be the set of actual clusters and *response* the set of clusters produced based on a distance metric for the same set of elements. A given pair of elements $(a, b)$ is considered as:

- *True Positive*, if $a$ and $b$ belong to the same cluster both in reference and response;
- *False Negative*, if $a$ and $b$ belong to the same cluster in reference, but to different clusters in response; and
- *False Positive*, if $a$ and $b$ belong to different clusters in reference, but the same cluster in response.

By applying this process to every pair of elements we can obtain the total number of *True Positives* (TP), *False Negatives* (FN) and *False Positives* (FP), based on which the *precision* and *recall* measures can be calculated as:

$$precision = \frac{TP}{TP+FP} \qquad recall = \frac{TP}{TP+FN}$$

*F-measure* is a balanced measure that takes into account both precision and recall, and is computed as:

$$F - measure = 2 * \frac{precision*recall}{precision+recall}$$

The predicted number of clusters $\hat{c}$ and $k_\alpha$ returned by the L Method and the Silhouette coefficient, along with the corresponding merge distance, F-measure, precision and recall values are shown for each examined web application in Tables III, IV, and V, respectively.

### C. Discussion

Observing Tables III, IV, and V one can see that the L method produced a diverse number of clusters for each distance metric. However, distance $d_C$ returned a $\hat{c}$ value that is closer to the actual number of clusters (and in the case of Garage the predicted number of clusters is exactly the same as the actual one) in all examined web applications. More specifically, for distance $d_C$ the L method returned 6 clusters for Google Maps (the actual number of clusters is 4), 19 clusters for Garage.ca (the actual number of clusters is 19), and 17 clusters for Kayak (the actual number of clusters is 16), while for distances $d_S$ and $d_L$ the returned $\hat{c}$ value is significantly different (larger or smaller) compared to the actual number of clusters.

Figure 3 clearly demonstrates that the evaluation graphs produced with $d_C$ as a distance metric have the closest resemblance with the typical graph required by the L method in order to effectively detect a knee. More specifically, by observing Figures 3b, 3e, and 3h we can see a distinct flat region of data points (on the right) followed by a sharply-increasing region of data points (on the left) in which the merge distances grow very rapidly. According to [11] this abrupt increase occurs when highly dissimilar clusters begin to be merged by the hierarchical clustering algorithm. The best number of clusters is located right before the sharp rise (when moving from right to left) on the evaluation graph. Thereby, the detection of the number of clusters from a knee region in graphs 3b, 3e, and 3h is more accurate. The results for distance metric $d_C$ demonstrate that this metric

Table III: Predicted number of clusters, F-measure, precision, and recall for Google Maps

| | L method | | | | | | Silhouette coefficient | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **distance** | **min** $RMSE_c$ | **merge dist.** | $\hat{c}$ | **F-measure** | **Precision** | **Recall** | $\alpha$ | **merge dist.** | $k_\alpha$ | **F-measure** | **Precision** | **Recall** |
| $d_S$ | 0.41 | 1.33 | 13 | 0.448 | 0.963 | 0.292 | 0.86 | 2.91 | 12 | 0.458 | 0.964 | 0.300 |
| $d_C$ | 0.18 | 3.33 | 6 | 0.832 | 0.985 | 0.721 | 0.79 | 4.41 | 5 | 0.930 | 0.937 | 0.924 |
| $d_L$ | 4.41 | 2.07 | 15 | 0.396 | 0.957 | 0.250 | 0.88 | 2.07 | 15 | 0.396 | 0.957 | 0.250 |
| **Actual number of clusters: 4**, $\hat{c}$: predicted number of clusters with L method, $k_\alpha$: predicted number of clusters with maximum Silhouette coefficient | | | | | | | | | | | | |

Table IV: Predicted number of clusters, F-measure, precision, and recall for Garage.ca

| | L method | | | | | | Silhouette coefficient | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **distance** | **min** $RMSE_c$ | **merge dist.** | $\hat{c}$ | **F-measure** | **Precision** | **Recall** | $\alpha$ | **merge dist.** | $k_\alpha$ | **F-measure** | **Precision** | **Recall** |
| $d_S$ | 4.90 | 8.52 | 25 | 0.861 | 0.972 | 0.774 | 0.67 | 20.90 | 15 | 0.852 | 0.834 | 0.872 |
| $d_C$ | 2.26 | 7.03 | 19 | 0.960 | 0.973 | 0.947 | 0.55 | 7.93 | 17 | 0.937 | 0.920 | 0.955 |
| $d_L$ | 2.23 | 11.47 | 14 | 0.856 | 0.840 | 0.872 | 0.78 | 14.59 | 13 | 0.854 | 0.837 | 0.872 |
| **Actual number of clusters: 19**, $\hat{c}$: predicted number of clusters with L method, $k_\alpha$: predicted number of clusters with maximum Silhouette coefficient | | | | | | | | | | | | |

Table V: Predicted number of clusters, F-measure, precision, and recall for Kayak

| | L method | | | | | | Silhouette coefficient | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **distance** | **min** $RMSE_c$ | **merge dist.** | $\hat{c}$ | **F-measure** | **Precision** | **Recall** | $\alpha$ | **merge dist.** | $k_\alpha$ | **F-measure** | **Precision** | **Recall** |
| $d_S$ | 2.74 | 7.79 | 24 | 0.832 | 1.00 | 0.713 | 0.81 | 51.88 | 15 | 0.994 | 0.996 | 0.992 |
| $d_C$ | 1.24 | 4.58 | 17 | 0.996 | 1.00 | 0.992 | 0.88 | 15.61 | 12 | 0.992 | 0.984 | 1.00 |
| $d_L$ | 2.72 | 4.86 | 29 | 0.770 | 1.00 | 0.626 | 0.80 | 20.64 | 21 | 0.820 | 1.00 | 0.694 |
| **Actual number of clusters: 16**, $\hat{c}$: predicted number of clusters with L method, $k_\alpha$: predicted number of clusters with maximum Silhouette coefficient | | | | | | | | | | | | |

produced well separated clusters for which the L method was able to determine an acceptable number of clusters very close to the number of clusters determined based on human perception. Also, as it can be observed from Tables III, IV, and V distance $d_C$, which handles composite changes in a special manner, achieved the best F-measure value in all examined web applications (0.832 for Google Maps, 0.960 for Garage, and 0.996 for Kayak) using the L method.

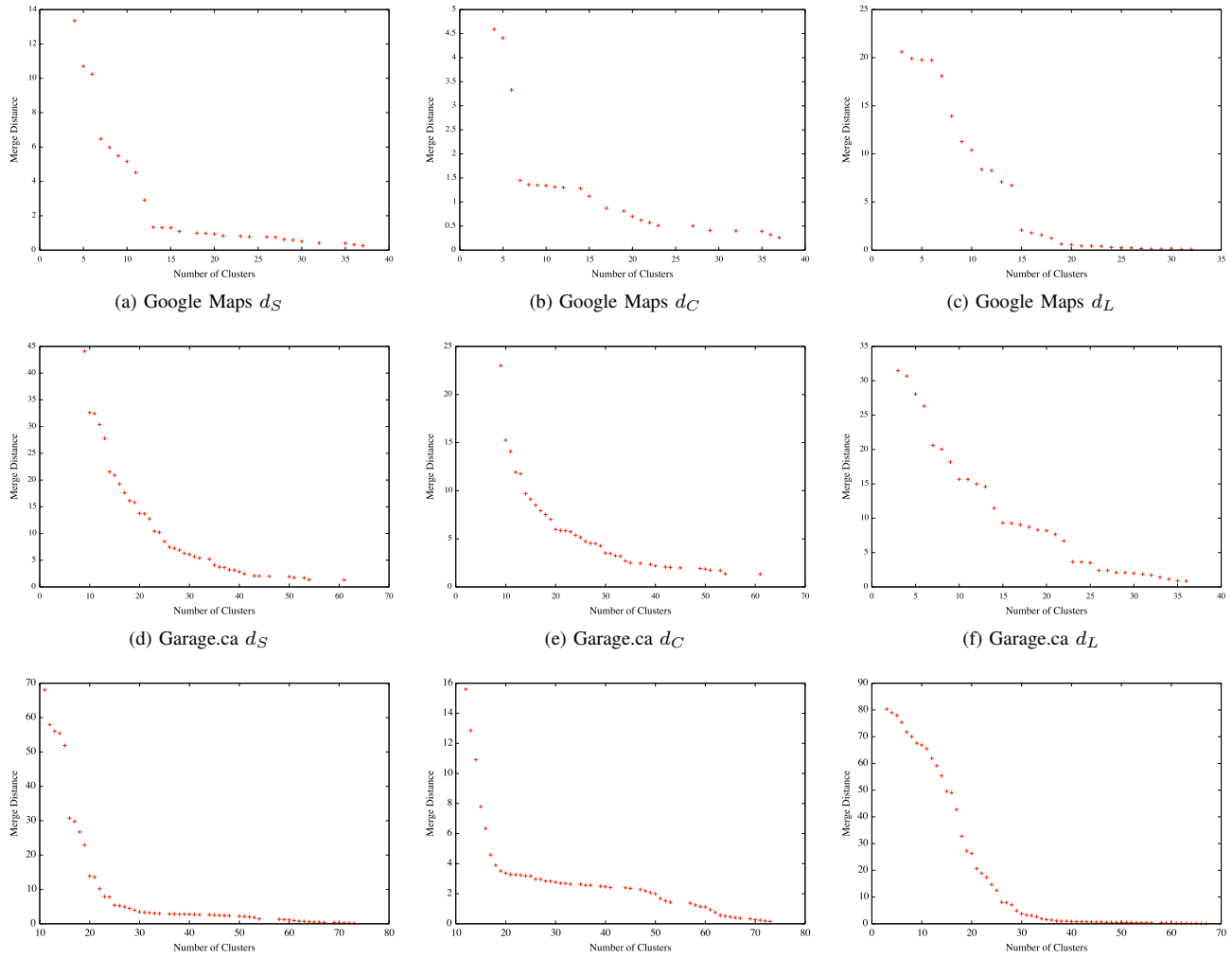
The evaluation graphs produced by the hierarchical clustering algorithm with distance metric $d_S$ (Figures 3a, 3d, and 3g) have a smoother transition between the flat and increasing regions of data points and thus the number of clusters returned by the L method for $d_S$ was not so precise. On the contrary, most of the evaluation graphs in Figures 3c, 3f and 3i, do not contain any obvious sharp transition, and therefore the L method could not determine a good number of clusters for the Levenshtein distance metric $d_L$. The knees on these evaluation graphs are ambiguous indicating that the distance metric used is not well-defined and produces clusters which are not clearly separated. The L method could not identify a reasonable number of clusters for Google Maps, and thus, the F-measure values for the corresponding partitions demonstrate poor clustering quality (0.448 for $d_S$ and 0.396 for $d_L$). For the other two applications, Garage and Kayak, the F-measure values for distances $d_S$ and $d_L$ are notably lower than the F-measure value for distance $d_C$.

The computation of the average Silhouette coefficient also resulted in a number of clusters that varies for each distance metric and application. However, according to the computed F-measure (0.930 for Google Maps, 0.937 for Garage, and 0.992 for Kayak), $d_C$ produced partitions of high quality for all three web applications, whereas the clustering quality of the partitions for distance metrics $d_S$ and $d_L$ is not stable and varies for each examined web application. More specifically, for Google Maps the predicted number of clusters for distance metrics $d_S$ and $d_L$ produced partitions with poor clustering results (F-measure is equal to 0.458 for $d_S$ and to 0.396 for $d_L$). For the other two web applications the F-measure values for distance $d_L$ are lower than the F-measure value obtained by distance $d_C$ (with the exception of $d_S$ in Kayak).

Clearly, the distance metric affects the quality of the clusters produced. In general, our distance metric $d_C$ produced good results using both methods for all three web applications (in contrast to the other examined distances). The fact that distance $d_C$ obtains the best results in a lower cut-off threshold compared to the other distance metrics implies that $d_C$ is a measure that in general produces smaller distances between the compared DOM trees.

### *D. Threats to validity*

An obvious threat to the internal validity of the conducted experiment is related with the determination of the *reference* set of clusters by the authors of the paper based on which the precision and recall of the examined distance metrics was extracted. In order to partially alleviate this threat, two authors of the paper grouped independently the screenshots of the web pages according to their visual perception of the




(a) Google Maps $d_S$ (b) Google Maps $d_C$ (c) Google Maps $d_L$

(d) Garage.ca $d_S$ (e) Garage.ca $d_C$ (f) Garage.ca $d_L$

(g) Kayak $d_S$ (h) Kayak $d_C$ (i) Kayak $d_L$

Figure 3: Evaluation graphs given as input to L method

feature that they provide. Next, they merged their results by reaching a common consensus in the cases of a different feature interpretation. In this way, we tried to eliminate the bias in the interpretation of the features. In general, the original clusters produced by the two authors had a very strong similarity ($\geq$ 90% based on Jaccard Index). In the cases where one of the authors grouped sub-clusters of features into a single higher-level feature, while the other one created a separate group for each of these lower-level features, the dissimilarities were eliminated by adopting the latter approach (a fine-grained decomposition of features).

A threat to the external validity of the experiment is the inability to generalize our findings beyond the examined web applications or distance metrics. Regarding the generalization of the results to other web applications, we have selected instances of web applications from three different domains, namely online shopping (Garage), map navigation and directions (Google Maps) and online travel booking (Kayak). More importantly, the selected web applications exhibit a variety in the consistency of their structure starting from one side with Google Maps which has a very stable and consistent user interface throughout the different features that it offers and going to the other side with Kayak which has a very diverse user interface with respect to the size and complexity of the generated DOM trees. This difference in the variation of size and depth between the generated DOM trees obviously lies in the different nature of the examined web applications.

## V. Conclusions and Future Work

In this paper, we proposed and evaluated a new distance metric for clustering similar DOM trees in Ajax-enabled web applications that reduces the impact of composite changes on the computation of structural similarity. The results of the evaluation on three different web applications have shown that the accuracy of the clustering results can be improved

over previous approaches that treat web pages as sequences of HTML tags (i.e., ignoring completely the tree structure of the web pages) and a simplified tree-edit-operation based distance metric that does not take into account composite changes.

Additionally, we applied two different methods, the L method [11] and the Silhouette coefficient [12], to investigate the ability of the examined distance metrics to automatically determine a number of clusters conforming to the actual number of features in the examined web applications. The results of the evaluation demonstrated that the process of extracting the features for a dynamic web application can be fully automated by employing the hierarchical clustering algorithm. The proposed distance metric $d_C$ produced partitions of clusters with high accuracy for all web applications. On the other hand, the other distance metrics did not work as well with the L method and Silhouette coefficient, producing clustering results with lower accuracy.

As future work we are planning to apply Latent Dirichlet allocation (LDA) [27] on the textual content of the extracted clusters of DOM trees in order to discover topics that could serve as names for the discovered features. An interesting difference with traditional documents containing plain text is that DOM trees contain hypertext where specific tags, such as headings, add emphasis to the surrounded text thus implying it has a greater importance compared to the rest of the text in the DOM tree.

## ACKNOWLEDGMENT

The authors would like to acknowledge the generous support of NSERC, iCORE, and IBM.

## REFERENCES

[1] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature location in source code: A taxonomy and survey," *Journal of Software Maintenance and Evolution: Research and Practice*, 2012.

[2] Y. Zou and M. Hung, "An approach for extracting workflows from e-commerce applications," in *Proceedings of the 14th IEEE International Conference on Program Comprehension*, ser. ICPC '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 127–136.

[3] Q. Zhang, R. Chen, and Y. Zou, "Reengineering user interfaces of e-commerce applications using business processes," in *Proceedings of the 22nd IEEE International Conference on Software Maintenance*, ser. ICSM '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 428–437.

[4] M. Jazayeri, "Some trends in web application development," in *2007 Future of Software Engineering*, ser. FOSE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 199–213.

[5] A. Mesbah, E. Bozdag, and A. van Deursen, "Crawling AJAX by inferring user interface state changes," in *Proceedings of the 2008 Eighth International Conference on Web Engineering*, ser. ICWE '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 122–134.

[6] F. Ricca and P. Tonella, "Using clustering to support the migration from static to dynamic web pages," in *Proceedings of the 11th IEEE International Workshop on Program Comprehension*, ser. IWPC '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 207–216.

[7] A. De Lucia, R. Francese, G. Scanniello, and G. Tortora, "Identifying cloned navigational patterns in web applications," *Journal of Web Engineering*, vol. 5, no. 2, pp. 150–174, 2006.

[8] G. A. Di Lucca, M. Di Penta, and A. R. Fasolino, "An approach to identify duplicated web pages," in *Proceedings of the 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment*, ser. COMPSAC '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 481–486.

[9] A. De Lucia, G. Scanniello, and G. Tortora, "Identifying similar pages in web applications using a competitive clustering algorithm." *Journal of Software Maintenance*, vol. 19, no. 5, pp. 281–296, 2007.

[10] A. De Lucia, M. Risi, G. Tortora, and G. Scanniello, "Towards automatic clustering of similar pages in web applications," in *Proceedings of the 11th IEEE International Symposium on Web Systems Evolution*, ser. WSE '09. IEEE Computer Society, 2009, pp. 99–108.

[11] S. Salvador and P. Chan, "Determining the number of clusters/segments in hierarchical clustering/segmentation algorithms," in *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence*, ser. ICTAI '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 576–584.

[12] P. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *J. Comput. Appl. Math.*, vol. 20, no. 1, pp. 53–65, Nov. 1987.

[13] A. Mesbah and A. van Deursen, "Migrating multi-page web applications to single-page ajax interfaces," in *Proceedings of the 11th European Conference on Software Maintenance and Reengineering*, ser. CSMR '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 181–190.

[14] G. A. Di Lucca, A. R. Fasolino, F. Pace, P. Tramontana, and U. De Carlini, "Comprehending web applications by a clustering based approach," in *Proceedings of the 10th International Workshop on Program Comprehension*, ser. IWPC '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 261–270.

[15] V. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions and Reversals," *Soviet Physics Doklady*, vol. 10, p. 707, 1966.

[16] R. Mikhaiel and E. Stroulia, "Accurate and efficient HTML differencing," in *Proceedings of the 13th IEEE International Workshop on Software Technology and Engineering Practice*, ser. STEP '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 163–172.

[17] M. Fokaefs, R. Mikhaiel, N. Tsantalis, E. Stroulia, and A. Lau, "An empirical study on web service evolution," in *Proceedings of the 2011 IEEE International Conference on Web Services*, ser. ICWS '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 49–56.

[18] K. Zhang and D. Shasha, "Simple fast algorithms for the editing distance between trees and related problems," *SIAM J. Comput.*, vol. 18, no. 6, pp. 1245–1262, Dec. 1989.

[19] E. W. Forgy, "Cluster analysis of multivariate data: efficiency vs interpretability of classifications," *Biometrics*, vol. 21, pp. 768–769, 1965.

[20] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley Symposium on Mathematical Statistics and Probability*. University of California Press, 1967, pp. 281–297.

[21] S. Dudoit and J. Fridlyand, "A prediction-based resampling method for estimating the number of clusters in a dataset." *Genome biology*, vol. 3, no. 7, Jun. 2002.

[22] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008.

[23] S. A. Elavarasi, J. Akilandeswari, and B. Sathiyabhama, "A survey on partition clustering algorithms," *International Journal of Enterprise Computing and Business Systems*, vol. 1, no. 1, pp. 1–14, 2011.

[24] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. of 2nd International Conference on Knowledge Discovery and Data Mining*, 1996, pp. 226–231.

[25] H.-P. Kriegel, P. Kröger, J. Sander, and A. Zimek, "Density-based clustering," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, no. 3, pp. 231–240, 2011.

[26] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM Comput. Surv.*, vol. 31, no. 3, pp. 264–323, Sep. 1999.

[27] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Mar. 2003.