Journal of Web Engineering, Vol. 8, No.4 (2009) 346-370
© Rinton Press

AN INVESTIGATION OF CLUSTERING ALGORITHMS IN THE IDENTIFICATION OF SIMILAR WEB PAGES

ANDREA DE LUCIA[+], MICHELE RISI[+], GIUSEPPE SCANNIELLO[*], GENOVEFFA TORTORA[+]

[+]*Dipartimento di Matematica e Informatica, University of Salerno, Italy*
{adelucia, mrisi, tortora}@unisa.it

[*]*Dipartimento di Matematica e Informatica, University of Basilicata, Italy*
giuseppe.scanniello@unibas.it

Received February 15, 2008
Revised August 6, 2009

In this paper we investigate the effect of using clustering algorithms in the reverse engineering field to identify pages that are similar either at the structural level or at the content level. To this end, we have used two instances of a general process that only differ for the measure used to compare web pages. In particular, two web pages at the structural level and at the content level are compared by using the Levenshtein edit distances and Latent Semantic Indexing, respectively. The static pages of two web applications and one static web site have been used to compare the results achieved by using the considered clustering algorithms both at the structural and content level. On these applications we generally achieved comparable results. However, the investigation has also suggested some heuristics to quickly identify the best partition of web pages into clusters among the possible partitions both at the structural and at the content level.

Keywords: clone analysis, clustering algorithms, latent semantic indexing, Levenshtein string edit distances, program comprehension, reverse engineering

1 Introduction

Web applications are playing an increasingly important role in the today society. Indeed, they represent the engine allowing distributed organizations to communicate, to share information with other companies, customers and providers, and to manage production and distribution. Similarly to traditional software systems, web applications change [6, 23]. It is due to internal and external factors, which generate new or modified system requirements. Differently from traditional software systems, web applications change much more quickly and due to the short time to market are often developed and/or evolved without adopting disciplined processes [25]. In case methodologies that anticipate changes and evolution are not applied, their comprehension, maintenance and evolution activities become challenging and time consuming [6, 8].

It is common opinion that the comprehension, maintenance, and evolution become even more complex in case software systems present similar or cloned code [3, 4, 10]. Thus, code clone detection can effectively support the improvement of the quality of traditional and web based applications during their maintenance and evolution. Various clone detection methods have been proposed in the literature [3, 5, 12, 13, 17, 43]. These methods and techniques can be used to support different activities during system evolution in general and web site evolution in particular; examples include reengineering, restructuring, and quality assessment [2, 19, 41, 42, 45].

Researchers have extensively studied clone detection for static web pages, written in HTML, or dynamic ones, written in ASP and JSP [7, 8, 13, 18, 20, 23, 44]. To identify cloned pages several completely different approaches based on the analysis of the page structure [17, 43], content [44, 45], or server side scripting code [10, 12] are available. For example, the identification of pages that are cloned or nearly identical at the structural level could be useful to reengineer them into a single dynamic page, while pages with similar content may be generalized by extracting the similar content from the pages and maintaining it in a separate file or in a database.

Basically, the majority of the clone detection approaches are based on clustering algorithms [12, 13, 14, 18, 41, 43, 44]. These approaches produce a high level view of applications in terms of software components and their relationships [12, 13, 18, 44]. For example, Di Lucca *et al.* in [18] present a clustering based approach to exemplify the navigational structure of a web application according to the typology and the topology of its interconnected components, thus making accessible detailed information exploding each cluster into sub clusters or into a single page. On the other hand, Ricca *et al.* [44] propose a semiautomatic approach based on clustering to group similar pages at the content level. Content similarity is computed using Natural Language Processing (NLP) techniques.

Generally, several non trivial issues have to be considered to avoid that a clustering based approach produces unsuitable results on web applications. First of all, a software engineer should choose the features of a web application to be considered in the clustering process (e.g., structure, content, or server side scripting code). Another relevant issue concerns the identification of a suitable measure to compare pages according to the chosen feature. The last issue to be addressed is the choice of the clustering algorithm to use. This choice may strongly condition the identification of similar or cloned pages whatever is the feature considered in the clustering based process.

This paper is based on our previous work [14] to further investigate the effect of using clustering algorithms employed in the reverse engineering field to group similar pages at the structural and content level within existing web applications. To this end, we have adopted two instances of the general process proposed in [13]. These instances only differ for the measure used to compare web pages. In particular, pages at the structural level are compared using the Levenshtein edit distance [36], while Latent Semantic Indexing (LSI) [16], an information retrieval technique, is adopted to get page dissimilarities at the content level. To group similar pages both instances use clustering algorithms belonging to the hierarchical and partitional categories [24]. Concerning the hierarchical category we consider the agglomerative [33] and divisive clustering algorithms [32], while within the partitional category we selected a variant [32] of k-means [24] algorithm and the Winner Takes All (WTA) [22] algorithm. To automate the clustering process a prototype of a supporting system has been also implemented. To investigate the effect of using the considered clustering algorithms and assess the system prototype, the results obtained on two web applications developed using JSP technology and one static web site have been compared. Indeed, such a comparison revealed that the considered clustering algorithms produced comparable results both at the structural and content level. Furthermore, the investigation presented in this paper has also shown that at the structural level WTA suggests heuristics to quickly identify the best partition of web pages into clusters among the possible partitions. At the content level using the k-means clustering algorithm heuristics to filter out surely bad configurations have also been identified and reported.

The remainder of the paper is organized as follows. Section 2 presents related work, while the process to group similar pages at the structural and content level together with the supporting tool are described in Section 3. Section 4 describes the considered clustering algorithms. The achieved results and their discussion are presented in Section 5 and Section 6, respectively. Finally, Section 7 concludes and outlines directions for future work.

2 Related Work

In the last decade, researchers studied extensively the problem of defining reverse engineering and analysis techniques for traditional [3, 4, 5, 48] and web based software systems [1, 8, 7, 20, 21, 42]. Ricca and Tonella in [42] propose the ReWeb tool to analyze the structure and the evolution of static web sites. They define a conceptual model for representing the structure of a web site. Several structural analyses relying on such a model, ranging from flow analysis to graph traversal algorithms and pattern matching, have been defined as well. Furthermore, the evolution of a web site is represented using colors as time indicators. These time indicators allow software engineers to determine how the original structure of a web site degrades during its lifecycle.

To comprehend the dynamic behavior of a web application the interactions among its software components has to be analyzed. Di Lucca *et al.* [20, 21] propose reverse engineering tools to extract the Conallen extension [11] of UML diagrams from existing web applications. The tools are based on static and dynamic analysis techniques. Antoniol *et al.* in [2] propose a methodology to reengineer static web sites. The methodology requires three phases: reverse engineering, restructuring, and forward engineering. In the reverse engineering phase the navigational structure of the web site is abstracted. Furthermore, the entities of the application domain and their relationships are identified and abstracted in ER+ diagram. The recovered model is then restructured and the forward engineering phase is performed using the RMM methodology [29].

Other authors focus on methods and tools to reverse engineer and comprehend large pieces of software, or whole legacy systems or existing web applications. A key activity in reverse engineering consists of gathering the software entities that compose the system into meaningful and independent groups. Such an activity is also known as clustering. In the past a large number of methods, approaches, and tools based on clustering algorithms have been proposed in the reverse engineering and comprehension of traditional software systems [1, 4, 28, 48] and web applications [7, 8, 12, 13, 14, 17, 18, 20, 42, 43]. For example, different authors have approached the problem of identifying cloned or similar web pages. Di Lucca *et al.* in [17] propose an approach that encodes the sequences of tags of HTML and ASP pages into strings and identify pairs of cloned pages at structural level by computing the Levenshtein string edit distance [36] between these strings. A pair of web pages is considered as perfect clone if their Levenshtein edit distance is zero. The authors also use metrics, such as lines of code and complexity metrics, to identify clones at the scripting code level. Similarly, De Lucia *et al.* [12] also use Levenshtein string edit distance to compute the similarity of two pages at structure, content, and scripting code level. Clones are characterized by a similarity threshold that ranges from 0%, for different pages, up to 100%, for identical pages. Let us note that also our approach uses the Levenshtein string edit distance to identify similar pages at the structural level.

Clustering algorithms are frequently adopted to identify clones in web applications. For example, Di Lucca *et al.* [18] propose a clustering method to decompose a web application into groups of functionally related components. The authors also define a coupling measure based on the number and type of links between pages and use an agglomerative hierarchical clustering algorithm to group pages together and simplify the navigational schema to make it more understandable. Ricca and Tonella in

[43] enhance the approach based on the Levenshtein edit distance proposed by Di Lucca *et al.* in [17] with a hierarchical clustering algorithm. In particular, they propose a semi automatic approach to identify clusters of similar pages to be generalized into a dynamic page. Each page is initially inserted into a different cluster and at each step clusters with minimal distance are merged thus producing a hierarchy of clusters. The clusters of cloned pages are selected by choosing a cut level in the hierarchy. However, the method only deals with the structures of static pages and do not consider the effect of using different clustering algorithms. Similarly, in [45] the same authors propose a semiautomatic approach to identify and align static HTML pages whose structure is the same and whose content is in different languages. Pages with similar structures are identified by adopting an agglomerative hierarchical clustering algorithm and using as basic distance measure the Levenshtein string edit distance. The aligned multilingual pages are merged into MLHTML pages. MLHTML is an extension of XHTML to develop multi lingual web sites. A different approach based on a competitive clustering algorithm is proposed in [13] to identify similar pages at the structural level. Page structures are encoded into strings and then the Levenshtein algorithm is used to achieve the distances between pairs of pages.

Ricca *et al.* in [44] describe the results of an empirical study to automate web site clustering considering the contents of their pages. Clustering is based on the similarity of keywords characterizing the text of each web page. Keywords are weighted so that more specific keywords receive a higher score. Indeed, the authors used Natural Language Processing techniques to weight each keyword, according to its relevance and specificity. Similar pages are then grouped together by adopting a hierarchical clustering algorithm.

Other methods have been proposed in the past to identify clones in web applications. For example, Calefato *et al.* [8] propose an approach to detect cloned functions within scripting code for static and dynamic web pages and extend the metric-based approach and the classification schema that Balazinska *et al.* suggest in [4]. Their approach exploits a pattern matching algorithm to compare scripting code fragments and is based on two steps: automatic selection of potential function clones based on homonym functions and size measures and visual inspection of selected script functions. This approach does not consider the problem of identifying cloned pages, as it does take into account neither the structure of web pages nor their content. The problem of identifying cloned pages from a structural point of view is not considered.

Rajapakse and Jarzabek in [41] analyzes the results achieved by applying different cloning approaches on web applications of different size and developed for different application domains by teams within different development environments. The study revealed that cloning equally affect small, medium or large web applications, and the number of clones increase over the time. The authors also showed that cloning could be even worse than that of traditional applications.

LSI has been employed in the past on different application domains [15, 35, 37, 39]. For example, Kuhn *et al.* in [35] describe a semantic clustering approach to identify components within existing software systems. The approach is language independent and tries to group source code containing similar terms in the comments. Different levels of abstraction to understand the semantics of the code (i.e., methods and classes) are considered. The analysis of the topic distribution has been addressed. This information retrieval technique has been also applied by Maletic and Marcus [37] to map high level concepts expressed in natural language to the relevant source code components implementing them and combined with structural information to cluster together software components for program comprehension. Another application of LSI concerns the comparison and the analysis of German literature texts [39]. Indeed, the author evaluates whether texts by the same author are alike and can be distinguished from the ones by different person. Texts by the same author are more alike and tend to form separate clusters. It has been also observed that the used information retrieval technique separates prose and poetry texts in two separate clusters.

3 A clustering process to identify similar pages

To identify the clustering algorithm that produces the best partition of web pages into clusters among the possible partitions either at the structural or at the content level we have used the general process proposed in [13] and depicted as an activity diagram with object flow in Figure 1. This process aims at identifying groups of similar pages using a suitable distance measure and any clustering algorithm. In particular, to compare pages according to the concern to analyze (e.g., structure, content, or server side scripting code) the *Page Distance Computation* phase has been defined. This phase is refined by the phases *Page Transformation* and *Computing Distance Matrix*. In the first phase representations of the pages to consider in the clustering process are produced. Successively, the Computing Distance Matrix phase employs the page representations to identify distance between all the pairs of pages of a given web application. Distances between the pairs of pages are used to produce a distance matrix, which is in turn provided as input to the phase *Grouping Similar Pages*. Different clustering algorithms can be used to implement this phase.

Two different instances of the general process depicted in Figure 1 have been used in the investigation presented here. In particular, the first instance has been defined to investigate the effect of using clustering algorithms widely employed in the past to identify similar pages at the structural level [12, 13, 17, 43, 45]. In this case, pages are compared using the Levenshtein string edit distance [36] (see Sections 3.1 for details). On the other hand, the second instance of the process has been defined to investigate the effect of using the same clustering algorithms in the identification of similar pages at the content level. A measure based on LSI (Latent Semantic Indexing) [16] has been properly defined and used to compare pages at the content level (see Sections 3.2 for details). The clustering algorithms selected for both the instances of the process belong to the hierarchical and partitional categories [24]. In particular, regarding the hierarchical category we consider the agglomerative [33] and divisive clustering algorithms [32], while within the partitional category we selected k-means [24] algorithm and the Winner Takes All (WTA) [22] algorithm. We describe these clustering algorithms in Section 4.

3.1 Identifying similar pages at the structural level

In order to cluster similar pages at the structural level, Page Transformation is in charge of producing string representations of the page structure of the static and dynamic pages. Page representation is produced through a depth-first traversal of the abstract syntax tree of the pages of a given web application. In our case each node of the syntax tree of a page is decorated with an HTML tag. An excerpt of an HTML page of a web application selected as case study and the corresponding syntax tree are shown in Figure 2. The excerpt of the tag sequence encoding the page structure is shown as well. Let us note that the syntax tree of a dynamic page differs from the syntax tree of a static page only for the presence of the server side scripting nodes.

Each HTML tag is encoded into different symbols of an alphabet before being concatenated into the string. The resulting string represents the structure of the considered web page. Encoding the tags enables a more precise computation of the distance of two pages with respect to just concatenating the HTML tags into strings. Furthermore, this encoding also improves the time required for the computation of the distance of two pages. It is important to point out that we have not considered the tag attributes in the strings encoding page structures. In the future we plan to investigate whether or not tag attributes influence the results.

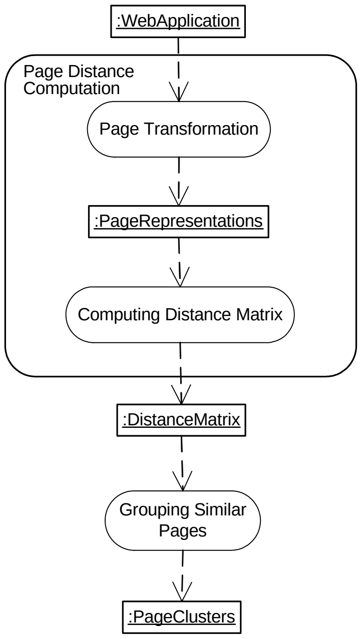
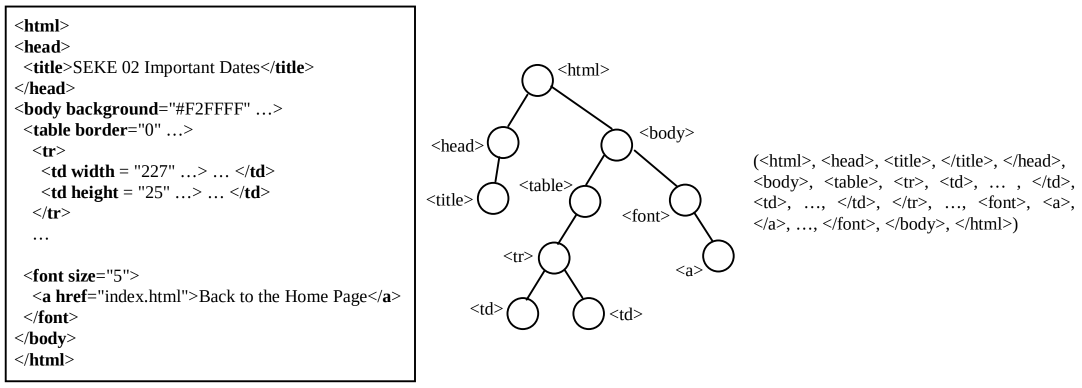


Figure 1 The clustering process

Once the strings encoding the page structures have been computed, the Levenshtein algorithm is used to get the distance between each pair of pages in the web application. This algorithm is one of the most known and employed dynamic programming algorithms. It has been originally proposed for string matching and then due to its flexibility it has also been used in several fields, such as code theory, phonetic distance, molecular biology, and speech recognition.



```
<html>
<head>
  <title>SEKE 02 Important Dates</title>
</head>
<body background="#F2FFFF" …>
  <table border="0" …>
    <tr>
      <td width = "227" …> … </td>
      <td height = "25" …> … </td>
    </tr>
    …

  <font size="5">
    <a href="index.html">Back to the Home Page</a>
  </font>
</body>
</html>
```

(<html>, <head>, <title>, </title>, </head>, <body>, <table>, <tr>, <td>, … , </td>, <td>, …, </td>, </tr>, …, <font>, <a>, </a>, …, </font>, </body>, </html>)

Figure 2 Abstract syntax tree and structure representation of a sample static page

The Levenshtein algorithm is based on the notion of edit operations (i.e., insert, delete, and replace) and consists of a set of rules that transform a source string into a target string. In particular, given two strings *x* and *y* the Levenshtein string edit distance is defined as the minimum number of insert, delete, and replace operations required to turn *x* into *y*. The operations insert and delete have 1 as cost, while 2 is the cost of the replace operation.

Figure 3 shows the tag sequences encoding the page structure of two sample pages, and how these structures have been encoded in the corresponding strings. The Levenshtein edit distance between these strings is shown as well.

Tag sequence:
(<html>, <head>, <title>, </title>, </head>, <body>, <font>, <a>, </a>, </font>, </body>, </html>)

Encoded string:
HDIEKBFALWYX

Tag sequence:
(<html>, <head>, <title>, </title>, </head>, <body>, <table>, <tr>, <td>, <font>, <a>, </a>, </font>, </td>, </tr>, </table>, </body>, </html>)

Encoded string:
HDIEKBTRCFALWNMQYX

Levenshtein string edit distance: 6

Figure 3 Structure representations of two sample static page, corresponding strings and the

To build the distance matrix (Computing Distance Matrix) of the pages of a given web application the Levenshtein string edit distance between all the possible pairs of pages is computed. The distance matrix is then provided as input to the phase Grouping Similar Pages in order to identify groups of similar pages at the structural level using the considered clustering algorithms.

3.2 Identifying similar pages at the content level

To compute the dissimilarity at the content level the Page Transformation phase extracts the static text by traversing the abstract syntax trees of the static and dynamic pages of a given web application. The extracted page content is then used in the phase Computing Distance Matrix to build the dissimilarity matrix of the web application. To this end, the term-by-content matrix is computed and then the Latent Semantic Indexing (LSI) [16], an extension of the Vector Space Model (VSM) [27], is used to get the concept space. LSI assumes that there is some underlying or latent structure in word usage that is partially obscured by variability in word choice, and uses statistical techniques to estimate this latent structure. The latent structure of a web application is obtained by applying a Singular Value Decomposition (SVD) [16] to the term-by-content matrix.

For purposes of intuition, SVD can be geometrically interpreted: terms and pages could be represented as vectors in the k space (i.e., the singular values of the dimensionality reduction of the latent structure of the contents) of the underlying concepts. Indeed, LSI allows a simple strategy for optimal approximate fit using smaller matrices and using only a subset of *k* concepts corresponding to the largest singular values in S. The choice of a "good" value of k is generally an open issue. In this investigation we calculate the number of singular values according to the Guttman-Kaiser criterion [26, 31].

Once the k space is obtained the dissimilarity between pairs of pages is computed using a measure based on the cosine between the vectors in the latent structure of the page content [16]. Indeed, the cosine between vectors is normalized between 0 (when the latent structure is the same) and 2 (when they have a different latent structure). In particular, given a web application and let *W* its latent structure and p_1 and p_2 two pages of this application, the dissimilarity measure is computed as follows:

$$d(p_1, p_2) = 1 - \cos(Vp_1, Vp_2)$$

where Vp_1 and Vp_2 are vectors corresponding to the pages p_1 and p_2 in the latent structure W. It is worth mentioning that the cosine similarity measure ranges from -1 (when the pages have a different latent structure) to 1 (when the latent structure is the same). Hence, the adopted measure is 0 in case the cosine between Vp_1 and Vp_2 is 1. Differently, this measure is 2 in case the angle between Vp_1 and Vp_2 is 180 degree, i.e., the cosine is -1. The defined dissimilarity measure differently from the Levenshtein string edit distance does not obey the triangle inequality rule. As result, this measure cannot be considered as a distance, but a semi-metric or more generally a dissimilarity measure. However, this does not influence the possibility of using clustering algorithms as Oudshoff *et al.* show in [40]. Let us note that the rationale for adopting the LSI technique depends on the fact that it has been successfully employed on different applications domains to identify semantic dependences among different entities (e.g., literature texts, software artifacts, source code, web pages, etc.) [15, 35, 37, 39].

3.3 Implementation

A prototype implementation of the process depicted in Figure 1 has been implemented in Java. To extract the structure and content of the pages of a given web application this prototype integrates an open source HTML parser written in Java (HTMLParser ver. 1.6 available under GPL license and downloaded from www.sourceforge.net/projects/htmlparser). The structure of each page is then encoded into a string as described in Section 3.1. To avoid parsing the pages more than once the software engineer can decide of storing the encoded structure and the content on the computer where the system prototype is executed.

To compute the distance matrix of the page structure the prototype integrates a C implementation of the Levenshtein string edit distance algorithm. The implementation of the Levenshtein algorithm is integrated using JNI (Java Native Interface). The distance matrix of the page content is computed using a LSI Engine implemented in R (available under GPL license from cran.r-project.org). As the computation of the distance/dissimilarity matrices could be expensive in terms of computation time the software engineer might store them before providing them as input to the different software components implementing the considered clustering algorithms.

To group similar pages at the structural and content level the prototype also integrates a MATLAB implementation of WTA and R implementations of the other clustering algorithms (available under GPL license from cran.r-project.org). The WTA implementation has been integrated using the JMatLink engine (available under GPL license from www.held-mueller.de/JMatLink). The R implementations of the clustering algorithms implemented in R have been integrated using the Rserve TCP/IP server. Also this server is available under GPL license and can be downloaded from http://rosuda.org/Rserve/down.shtml. Even, the used LSI Engine has been integrated within the prototype using Rserve. Further details on the prototype are not provided as its architecture is based on the prototype discussed in [13].

4 Clustering algorithms

Clustering is the unsupervised classification of entities or observations into groups (clusters). To compare observations and then group them a proximity measure has to be carefully chosen. It is most common to calculate the dissimilarity between two entities using a distance or dissimilarity measure defined according to the observation to be grouped into clusters [30]. As mentioned above, in this paper we consider three variants of the agglomerative hierarchical clustering algorithm and a divisive clustering algorithm. We also investigate the effect of using two well known partitional clustering algorithms: k-means [38] and Winner Takes All (WTA) [22]. In the following subsections the selected clustering algorithms are described.

4.1 Agglomerative hierarchical clustering

The adopted agglomerative hierarchical clustering algorithm [33] begins with each page in a distinct cluster, and then hierarchically merges clusters together until a stopping criterion is satisfied. To update the distance/dissimilarity between pairs of clusters several methods can be applied. In this paper we consider the most popular methods: single-link, complete-link, and average-link [33]. In the single-link method, the distance/dissimilarity between two clusters is the minimum of the distances/dissimilarities between all pairs of pages in the clusters (one page from the first cluster, the other from the second), while in the complete-link method, the distance/dissimilarity between two clusters is the maximum of the distances/dissimilarities between all pairs of pages. In the average-link method, the distance/dissimilarity between two clusters is the mean value of the distances/dissimilarities between all the pairs of pages in the clusters.

To illustrate the arrangement of the clusters produced by an agglomerative hierarchical clustering algorithm a dendrogram is frequently used (see Figure 4). Dendrograms are tree based representations particularly suitable to represent the nested groups of pages and the levels at which groups change. To obtain different clusters, dendrograms can be cut at different levels. Each cut level represents a tuning value of our approach. Figure 4 shows a dendrogram sample and a cut level enabling us to identify six clusters. In this example the dissimilarity between pairs of clusters is updated using the complete-link method.

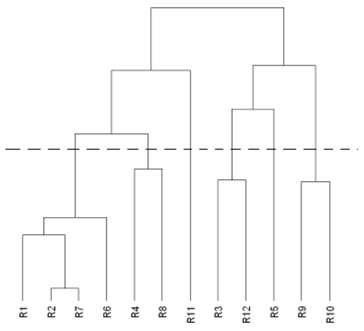


Figure 4 A dendrogram produced by an agglomerative hierarchical clustering algorithm

4.2 Divisive hierarchical clustering

In the considered divisive hierarchical clustering algorithm, clusters are divided until each page is placed in a distinct cluster. The choice of the divisive step represents our tuning values. At each step, the cluster with the largest diameter is selected, where the diameter of a cluster is defined as largest distance/dissimilarity between any pairs of pages in the cluster. To divide the selected cluster, the algorithm first looks for its most disparate page characterized by the largest average distance/dissimilarity to the other pages. The disparate page is then used to divide the cluster in two new clusters. In the first cluster there will be the pages close to the disparate page, while the remaining pages will be placed in the second cluster. The closeness of a cluster to the disparate page is computed considering a divisive coefficient. For each page i, we denote *d(i)* as the diameter of the last cluster to which i belongs (before being split off as a single page), divided by the diameter of the pages of a web application. The divisive coefficient is the average of all *1 − d(i)*. Note that the dendrogram representation is also suitable to represent the different iterations of a divisive hierarchical clustering algorithm.

4.3 K-means

The K-means algorithm [38] is used to classify pages of a given data set through a priori fixed number of disjoint clusters (i.e., our tuning value). The main idea is to define a centroid for each cluster of pages to identify. K-means iteratively refines the initial centroids, minimizing the average distance/dissimilarity, i.e. maximizing the similarity, of pages to their closest centroids. At each iteration, the clusters are built by assigning each cluster to the closest centroids. We adopt here a variant of the k-means that minimizes the sum of the distances/dissimilarity instead of the sum of squared Euclidean distances. The considered variant is composed of two phases: *building* and *swap*. In the building phase the algorithm looks for a good initial set of centroids. Note that a different location of the centroids causes the identification of different clusters. To avoid biasing the results of the clustering process, centroids have to be placed as much as possible far away from each other. In the swap phase the algorithm finds a local minimum for the objective function, that is, a solution such that there is no single switch of a centroid that minimize the sum of the distances/dissimilarity.

4.4 Winner takes all

One of the widespread partitional approaches to cluster observations from the statistical recognition perspective is WTA (Winner Takes All). It is an Artificial Neural Network (ANN) [22] clustering algorithm aiming at grouping and representing similar observations by a singular unit also known as neuron. We adopt as ANN the Self-Organizing feature Map (SOM) that in literature someone calls Kohonen feature map [34].

The SOM network is based on a two dimensional grid of neurons, whose weights are continuously adapted to the training set of the clustering algorithm, i.e. the vectors of the distances between the pages of a web application. The number of neurons is provided as input to the SOM network and represents the number of clusters that the algorithm should identify. This number is a-priori chosen by the software engineer and represents our tuning value. To avoid analyzing the dataset before running the algorithm the initial positions of the neurons on the SOM lattice are randomly assigned.

The neuron with weight vector most similar to the input vector is called the Best Matching Unit (BMU) or winner. The weights of the BMU and neurons close to it in the SOM lattice are adjusted towards the input vector. The magnitude of the change decreases with time and is smaller for neurons physically far away from the BMU. This process is also known as training process and is repeated over and over for each page. The training process is concluded when either the neurons did not change their position on the SOM lattice or a termination threshold (i.e., number of epochs) for the iteration has been reached. The network winds up associating the neurons to groups of similar pages in the input space. Whether these groups can be named, the names can be attached to the associated neurons in the trained net. The named neurons represent the clusters of static/dynamic web pages identified by the SOM network. The pages in the input space are associated to the closest neuron. It is possible that at the end of the process, some neurons do not have any page associated, i.e. they result in empty clusters. In other words, the number of identified clusters can be lower than the number of neurons in the network (i.e., the number of expected clusters).

5 Case Study

To investigate the effect of using the selected clustering algorithms we have considered the static pages of three applications, two were web applications implemented using J2EE technologies and one was a static web site. Regarding the web applications, we considered the official web site of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE 02) and the SRA (Student Route Analysis) web application. Finally, as static web site we analyzed EC (EasyClinic). At the structural level the clustering algorithms have been compared considering static and dynamic pages separately. Regarding the investigation at the content level, we have only considered static pages as little static text was contained within the dynamic pages of the selected web applications.

5.1 Selected Web Applications

SEKE 02 was used to support the organizers and the academic community for paper submission, refereeing, and conference registration. This application was composed of 1268 files distributed in 63 folders. Among these files the static and dynamic pages were 157 pages; which were included in one directory without any meaningful classification. SEKE 02 similarly to the other analyzed applications also contained some other files, such as images, java applets, java classes, logos, etc.

SRA was devised to provide the students in Politics Science at University of Salerno with statistics and information on their academic carrier. To make information also available to visiting students the application presents information also in English language. The SRA web application was composed of 380 files distributed in 45 folders according to a meaningful classification. Overall, the SRA web application was composed of 24 HTML pages and 45 JSP pages.

A group of master students in Computer Science at the University of Salerno, who attended the Software Engineering course, implemented the EC web site to share the software artifacts produced during the development process of the EasyClinic software system. Thus, this site provided information on the static and dynamic models of the requirement analysis phase, the class descriptions of the object model produced in the object design phase, and the description of the test cases. EC was constituted of 321 files distributed in 4 folders. Depending on the phase of the development process the static pages were properly grouped. Furthermore, a static page used to navigate the web application and not considered in the page analysis was also implemented. Let us note that the EC pages had nearly the same structure. This was due to the fact that EC was essentially developed to share information among the team members. This resulted in static pages with the same structure. Hence, we only analyzed the pages from the content point of view. Statistics of the web site and the web applications considered in the case study are summarized in Table 1.

	SEKE 02	**SRA**	**EC**
Number of files	1268	380	321
Number of HTML pages	49	24	161
Number of JSP pages	108	45	-
Number of folders	63	45	4

Table 1. Statistics on the analyzed static and dynamic web sites

It is worth noting that the investigation presented in this paper is focused on all the possible combinations of the factors: page type (i.e., static or dynamic) and analysis level (i.e., structural and content). Table 2 summarizes how the static and dynamic pages of the chosen web site and web applications have been analyzed with respect to their structural and content. For example, the static pages of SEKE 02 have been analyzed at the structure and content level as the corresponding entry (*Structure/Content*) of the table shows.

Table 3 and Table 4 show statistics on the pages of the web site and the two web applications analyzed in the proposed investigation. Descriptive statistics have been provided according to the considered combinations of the factors. In particular, Table 3 shows descriptive statistics on the pages of the web application SEKE 02 and SRA with respect to the page structures, while descriptive statistics on the words within the static page content of SEKE 02, SRA, and EC are reported in Table 4. The first row from each table shows the minimum number of tags and words of the smallest page of a given web application, while the maximum number of tags and words of the largest page are reported in the second row. The mean number of tags and words for page are reported on the third row. The median and the standard deviation of the number of tags and words for each web application are shown in the fourth and fifth rows, respectively. These statistics provide relevant information on the size and the variability of the number of tags and words of the analyzed web pages. The number of pairs of similar pages that have been manually recognized as similar at the structural and content level are reported in the sixth rows. The last rows show the similarity density measure, which is defined as the ratio between the total number of pairs of pages that have been recognized as similar over the number of possible pairs of similar pages in a web application.

SEKE 02		**SRA**		**EC**
HTML	**JSP**	**HTML**	**JSP**	**HTML**
Structure/Content	*Structure*	*Structure/Content*	*Structure*	*Content*

Table 2. Possible combinations of the factors on which our investigation is focused on

	SEKE 02		**SRA**	
	HTML	**JSP**	**HTML**	**JSP**
Minimum number of tags	23	8	12	10
Maximum number of tags	7879	1295	60	3239
Mean number of tags	1266.6	533.6	39.7	354.6
Median	939	583	47	260
Standard deviation	1640.9	307	17.2	508.7
Manually detected pairs of pages similar at the structural level	19	87	48	15
Similarity density measure at the structural level	1.6%	1.5%	17.4%	1.5%

Table 3. Descriptive statistics on the page structures

	SEKE 02	SRA	EC
Minimum number of words	39	8	41
Maximum number of words	2998	102	770
Mean number of words	427.8	51.4	153
Median	296	46	141.5
Standard deviation	567.2	28.9	88.5
Manually detected pairs of pages similar at the content level	18	14	1005
Similarity density measure at the content level	1.5%	5%	7.9%

Table 4. Descriptive statistics on the page content

5.2 Assessing the Results

Different methods have been proposed in the past to analyze the results produced using a clustering based approach [13, 44, 46]. To compare the results achieved by applying the different clustering algorithms we have used here two well known measures, namely precision and recall. In particular, if A is the set of pairs of similar pages identified by the tool and B is the set of actual pairs of similar pages, the recall and the precision are defined as follows:

$$recall = \frac{|A \cap B|}{|B|} \qquad precision = \frac{|A \cap B|}{|A|}$$

The recall measure can be interpreted as the percentage of similar pages that the tool prototype has correctly identified. On the other hand, the precision measure represents the percentage of similar pages identified by the tool prototype that are actually correct.

In order to compute precision and recall, the actual groups of similar pages have been manually produced by two experts who were not in the development team of the tool prototype. The two experts worked independently and iterated until they reached an agreement. Note that pages were considered similar at the structural level in case they had a very similar layout (e.g., the same navigational menu, graphic header, and colophon). On the other hand, pages were grouped into clusters of similar pages at the content level in case they contained text that was semantically similar.

To compute precision and recall the groups of similar pages manually identified and the groups of pages identified by the prototype are turned into pairs of similar pages. For instance, if (P_1, P_2, P_3) is a cluster of similar pages, the following are the corresponding pairs of clones:

(P_1, P_2,) (P_1, P_3) (P_2, P_3)

By definition the clone relation is reflexive (a page is a clone of itself), symmetric (if a page P_1 is a clone of a page P_2, then P_2 is a clone of P_1), and transitive (if a page P_1 is a clone of a page P_2 and P_2 is a clone of a page P_3, then P_1 is a clone of P_3). It is worth mentioning that clones manually identified are reported in a gold matrix, while those identified by the tool are reported in result matrix. Thus, different result matrices and precision and recall values are obtained by applying different tuning values on each investigated clustering algorithm.

The evaluation of precision and recall is not straightforward. Thus, to better assess the results obtained by applying the different tuning values on each clustering algorithm we also considered the trade-off between the precision and recall values. In particular, we used the F-measure, which was defined as the harmonic mean of precision and recall:

$$2 * \frac{precision * recall}{precision + recall}$$

Consequently, given a clustering algorithm and a web application, we defined as *trade-off configuration* the tuning configuration that allows achieving the largest F-measure value. In particular, for WTA and k-means we considered all the configurations ranging from the number of static pages to 1 and computed for each configuration the precision, the recall, and the F-measure. The number of expected clusters that enabled us to get the larger F-measure value represented the trade-off configuration. As configurations of the three different versions of the hierarchical clustering algorithm we considered all the cutting levels of the corresponding dendrograms. For the divisive clustering algorithm we considered as configurations all the cutting levels and computed for each configuration the precision, the recall, and the F-measure. Regarding the three variants of the hierarchical clustering algorithms and the divisive clustering algorithm the cutting levels produced the larger F-measure values represented the trade-off configurations. Let us note that the measures precision, recall, and F-measure assume values ranging between 0 and 1.

5.3 Results at the structural level

Table 5.a and 5.b report on the trade-off configurations of the static and dynamic pages of SEKE 02, respectively. In particular, the first columns contain the name of the clustering algorithms, while the trade-off configurations are reported in the second columns. The second columns for WTA also present the number of expected clusters (i.e., the first number) and the number of clusters that the algorithm actually identified (i.e., the second number). The precision and recall values are shown in the third and fourth columns, respectively. Finally, the F-measures of the presented configurations are contained in the last columns.

The precision and recall values corresponding to the trade-off configurations of the SEKE 02 static pages (see Table 5.a) revealed that the k-means clustering algorithm produces better results (40 is the trade-off configuration). The trade-off configurations of the divisive clustering algorithm and the hierarchical clustering algorithms produced 38 clusters. On the other hand, 31 is the trade-off configuration of the WTA clustering algorithm. Note that the WTA clustering algorithm produced worse results (i.e. the precision and recall values were 0.523 and 0.578, respectively) than the other clustering algorithms.

Table 5.b shows that the better F-measure value (i.e., 0.695) was obtained by using the divisive hierarchical clustering algorithm. Indeed, all the clustering algorithms except WTA produced comparable F-measure values. However, WTA enabled us to get a better recall value. Finally, the number of clusters identified by the WTA algorithm is lower than the number of clusters identified by the other algorithms and the number of identified clusters coincides with the number of expected clusters.

The results achieved on the static pages of SRA are shown in Table 6.a. The divisive clustering algorithm produced better results (i.e. the precision and recall values were 0.959 and 0.979, respectively). Similar results were achieved by employing the three variants of the agglomerative clustering algorithm and WTA (the precision and recall values were 0.921 and 0.974, respectively). Differently, the precision value was 1 and the recall value was 0.812 when k-means was used. Regarding the WTA clustering algorithm the number of expected and identified clusters coincides. The results of the trade-off configurations on the dynamic pages are shown in Table 6.b. Let us note that the clustering algorithms generally produced bad results. This is could be due to the low number of dynamic pages that are actual clones at the structural level.

	Clusters	**Prec.**	**Recall**	**F-meas.**
Agglom. complete	38	0.565	0.684	0.619
Agglom. single	38	0.565	0.684	0.619
Agglom. average	38	0.565	0.684	0.619
Divisive	38	0.565	0.684	0.619
k-means	40	0.923	0.631	0.750
WTA	31/31	0.523	0.578	0.549

(a)

	Clusters	**Prec**	**Recall**	**F-meas.**
Agglom. complete	90	0.877	0.574	0.694
Agglom. single	90	0.877	0.574	0.694
Agglom. average	90	0.877	0.574	0.694
Divisive	87	0.820	0.603	0.695
k-means	86	0.781	0.574	0.662
WTA	57/57	0.552	0.660	0.601

(b)

Table 5. Results obtained by using the trade-off configurations on the static (a) and dynamic pages (b) of SEKE 02 at the structural level

	Clusters	**Prec.**	**Recall**	**F-meas.**
Agglom. complete	6	0.921	0.979	0.949
Agglom. single	6	0.921	0.979	0.949
Agglom. average	6	0.921	0.979	0.949
Divisive	7	0.959	0.979	0.969
k-means	9	1	0.812	0.896
WTA	6/6	0.921	0.979	0.949

(a)

	Clusters	**Prec.**	**Recall**	**F-meas.**
Agglom. complete	33	0.437	0.466	0.451
Agglom. single	37	0.454	0.333	0.384
Agglom. average	32	0.35	0.466	0.400
Divisive	33	0.437	0.466	0.451
k-means	31	0.208	0.333	0.256
WTA	27/26	0.206	0.4	0.272

(b)

Table 6. Results obtained by using the trade-off configurations on the static (a) and dynamic pages (b) of SRA at the structural level

5.4 Results at the content level

The achieved results at the content level on the static pages of SEKE 02 are presented in Table 7. The agglomerative hierarchal clustering algorithms based on the methods average and single-link produced better results than the other clustering algorithms. In particular, 0.657 and 0.638 are the F-measure values obtained by applying the hierarchical clustering algorithm based on the methods average and single-link, respectively. Among the considered clustering algorithms WTA produced the worst results. Furthermore, the number of expected clusters at the content level coincides with the number of identified cluster. We observed that on each considered web application the number of expected and identified clusters always coincides. Hence, in the following of this section we do not distinguish between expected and identified clusters.

	Clusters	**Precision**	**Recall**	**F-meas.**
Agglom. complete	33	0.633	0.527	0.575
Agglom. single	35	0.638	0.638	0.638
Agglom. average	35	0.676	0.638	0.657
Divisive	36	0.633	0.527	0.575
k-means	39	0.692	0.5	0.581
WTA	28	0.423	0.611	0.5

Table 7. Results obtained by using the trade-off configurations on SEKE 02 at the content level

	Clusters	**Precision**	**Recall**	**F-meas.**
Agglom. complete	11	0.366	0.785	0.5
Agglom. single	9	0.218	1	0.358
Agglom. average	11	0.366	0.785	0.5
Divisive	11	0.366	0.785	0.5
k-means	12	0.523	0.785	0.628
WTA	18	0.888	0.571	0.695

Table 8. Results obtained by using the trade-off configurations on the static pages of SRA at the content level

On the SRA pages the clustering algorithms produced different results as shown in Table 8. The clustering algorithm enabling us to achieve the larger F-measure value (i.e., 0.695) was WTA. A similar F-measure value (i.e., 0.628) was produced by k-means. The remaining algorithms except the agglomerative clustering algorithm based on the single-link method produced the same results. In particular, 0.5 is the F-measure obtained by using these clustering algorithms. Differently, the agglomerative clustering algorithm based on the single-link method produced 0.358 as F-measure value.

	Clusters	Precision	Recall	F-meas.
Agglom. complete	36	0.611	0.357	0.45
Agglom. single	58	0.537	0.361	0.432
Agglom. average	77	0.964	0.329	0.447
Divisive	42	0.674	0.339	0.451
k-means	19	0.493	0.353	0.411
WTA	79	0.438	0.373	0.403

Table 9. Results obtained by using the trade-off configurations on EC at the content level

The F-measure values obtained on the EC web site were nearly the same (see Table 9). Indeed, the divisive clustering algorithm produced better results. In particular, it enabled us to get 0.451 as F-measure value, and 0.674 and 0.339 as precision and recall values, respectively. Similar F-measure values were obtained by applying the three variants of the agglomerative clustering algorithms. Let us also note that the agglomerative clustering algorithm based on the average-link method produced a better precision value and a worse recall value with respect to the same clustering algorithm based on the single-link and complete-link methods. Worse results were achieved on EC using the clustering algorithms k-means and WTA.

6 Discussion

In the following subsections the obtained results and remarks are discussed.

6.1 General findings

Information on the selected web applications has been obtained by interviewing their developers and inspecting their source code. In case the developers were not available the source code has been manually analyzed as it represented the only viable alternative. The SEKE 02 web application has been developed without using systematic development methodology and its requirements were unclear and changing. This web application also presented a considerable number of unreachable pages, so its navigational schema appeared as a set of disconnected graphs. Differently, the SRA web application has been developed using a systematic development process and Jakarta Struts as development framework. Finally, any methodology has been adopted to develop the EC web site. This was due to the fact that EC was intended to share information among a group of students. The selected web applications are quite different in terms of size, complexity, quality, and used development approach. However, the analysis of the results achieved at the structural level has revealed that on each selected web application all the clustering algorithms often produced comparable results. Furthermore, the three variants of the agglomerative clustering algorithms always produced similar results, thus getting frequently nearly identical precision and recall values.

Regarding the results achieved by employing the clustering algorithms to identify pages that are similar at the content level, we observed that WTA generally produced worse results except in case when it has been employed on a multilingual web application (i.e. SRA). In fact, on the static pages of SRA the better F-measure value was achieved when WTA was applied. Furthermore, as compared to the other clustering algorithms a high precision value (i.e., 0.888) was achieved (see Table 8). WTA also produced worse results when used on web applications with a low density of pairs that have been manually recognized as similar, e.g., the SEKE 02 web application (see Table 1). Also, at the content level the three variants of the agglomerative clustering algorithms produced similar results, thus getting frequently similar values of precision and recall. Indeed, on the smaller web applications identical values of precision and recall were achieved. It is worth noting that we generally obtained worse results in case the selected clustering algorithms have been used to group similar pages at the content level with respect to their employment in the identification of similar pages at the structural level.

6.2 Analyzing pages that are similar at the structural and content level

It is easy to imagine cases where structurally similar pages have similar content and vice versa. Therefore, it would be interesting to see in how many cases pages that are similar at the structural level match or not with the pages that are similar at the content level. Similarly, it would be interesting to consider the matching between pages that are identified by the tool as similar at the content level and pages that have been identified as similar at the structural level. We have conducted this further investigation on the static pages of SEKE 02 and SRA. This was due to the fact that the static pages of these web applications have been analyzed both at the structural and content level.

	SEKE 02		**SRA**	
	Pairs identified by the tool	**Actual pairs identified by the tool**	**Pairs identified by the tool**	**Actual pairs identified by the tool**
Agglom. complete	4	4	13	9
Agglom. single	6	6	14	9
Agglom. average	6	6	13	9
Divisive	4	4	13	9
k-means	6	6	12	9
WTA	5	5	11	9

Table 10. Numbers of pairs of pages similar both at the structural and content level

We first compared the pages that have been identified by the tool as similar both at the structural and content level using the trade-off configurations for each chosen clustering algorithms. Indeed, we first intersect these pairs and then analyzed them to recognize the actual ones. Table 10 summarizes the results achieved by using the chosen clustering algorithms on SEKE 02 and SRA. Let us note that the total numbers of pairs of pages that are actually similar both at structural and content level for SEKE 02 and SRA are 8 and 9, respectively.

Note that on SEKE 02 all the pairs of pages identified by the tool are actual pairs. However, the tool was not able to identify all the pairs of pages that are actually similar both at the structural and content level. On the other hand, on SRA the tool prototype was able to identify all the pairs of pages that have been previously manually detected. Nevertheless, the tool has identified a number of pairs of pages larger than the ones that are actually similar both at the structural and content level. A further investigation of these pairs of pages revealed that the majority of these pairs were similar at the structural and/or content level.

It may also happen that pairs of pages similar at the structural level may be not similar at the content level and vice versa. In particular, let *S* the set of pairs identified by the tool as similar at the structural level and *C* the set of pairs that the tool identified as similar at the content level we have analyzed the pairs of pages obtained by performing S-C and C-S. Table 11 shows the number of pairs of pages that are similar at the structural level and are not similar at the content level (i.e., S-C). The number of pairs of pages similar at the content level and not similar at the structural level (i.e., C-S) is shown as well. We can observe that SEKE 02 as compared to SRA has a larger number of pairs of pages that are similar both at the structural and content level.

6.3 Heuristics to identify trade-off configurations

The identification of a trade-off configuration on real and large web applications is not straightforward in practice and is generally expensive and time consuming [13, 41, 44]. This is because groups of pages have to be manually assessed to compute precision and recall. This means that all the pairs of similar pages have to be manually identified.

	SEKE 02		**SRA**	
	Structural vs Content	**Content vs Structural**	**Structural vs Content**	**Content vs Structural**
Agglom. complete	19	11	38	17
Agglom. single	17	12	37	50
Agglom. average	17	11	38	17
Divisive	19	11	36	17
k-means	7	16	27	9
WTA	16	9	38	8

Table 11. Numbers of pairs of pages that are similar at the structural and are not similar at content level and vice versa

Guidelines to automatically filter out surely bad tuning configurations and to get more quickly the trade-off configuration should be devised. In particular, at the structural at the content levels the guidelines have been defined on the clustering algorithms WTA and k-means, respectively. To better clarify the guidelines at the structural level let us define the break-even configuration.

Definition 1: the break-even configuration is the larger tuning value such that WTA identifies non-empty clusters.

We have observed that on the web site and web applications used as case study, the WTA clustering algorithm produced similar results using both the break-even and the trade-off configurations. Differently from the trade-off configuration, the break-even configuration can be easily computed by trying all possible configurations, ranging from the page number of a web application to 1, and looking for the configuration where the number of empty clusters becomes 0 for the first time. Once this configuration is identified, the software engineer should analyze the configurations close to it. In this way, only a relatively small number of configurations have to be analyzed, thus reducing the effort required to approach the trade-off configuration.

Table 12 reports on the trade-off and break-even configurations obtained by using WTA. In particular, the first row shows the name of the analyzed web applications, while the second row reports the page type (i.e., HTML and JSP). The third and the fourth rows contain the trade-off and the break-even configurations, respectively. For example, this table shows that on the static pages of SEKE 02 the break-even and the trade-off configurations are nearly identical. On the other hand, the break-even and trade-off configurations coincide on the structure of the SRA static web pages.

We also observed that a larger difference between the break-even and trade-off configurations was obtained in case a web application had a low similarity density value (for example on the dynamic pages of SRA). However, maintenance operations should be not required in case a web application has a low number of static and/or dynamic pages that are recognized as true clones.

	SEKE 02		**SRA**	
	HTML	**JSP**	**HTML**	**JSP**
Trade-off configuration	31	57	6	27
Break-even configuration	32	61	6	18

Table 12. Trade-off and break-even configurations of WTA on the analyzed web applications

To comprehend the guidelines to prune surely bad configuration at the content level the definition of the no-single cluster configuration is needed.

Definition 2: the no-single cluster configuration is the larger tuning value such that all the clusters identified by the k-means algorithm contain at least two pages.

As it is easy to imagine, the no-single cluster configuration can be simply computed by trying all possible configurations (ranging from the number of pages divided by two to 1). The first configuration where all the identified clusters contain at least two pages is the no-single cluster configuration. Trade-off and no-single cluster configurations of the considered web based applications are reported in Table 13. In particular, the first row shows the name of the analyzed web applications. The second and the third rows contain the trade-off and the no-single cluster configurations, respectively.

Table 13 shows that the no-single cluster and the break-even configurations of EC are nearly the same. On the other hand, the no-single cluster and the break-even configurations of the remaining applications are different. This could be due to the fact that EC had a larger similarity density (i.e., 7.9%) as compared to the remaining web based applications (see Table 4). However, on SEKE 02 and SRA the trade-off configurations are always larger than the no-single cluster configurations. This means that in case of web applications with a low similarity density, the software engineer has to analyze a smaller set of possible configurations as compared to all the possible configurations.

A further data analysis showed that on SEKE 02 a better recall value was obtained when the no-single cluster configuration was used (i.e., 0.889). However, less effort is generally required to remove false clones with respect to identify pages that are true clones. Regarding SRA, we observed that the recall value was 0.875 using both the trade-off and the no-single cluster configurations. On the other hand, a smaller precision (i.e., 0.22) value was obtained using the no-single cluster configuration: a larger number of false clones were detected. This could be considered not an issue as the elimination of false positive is less expensive than the detection of pages that are actual clones. This result also provides an insight on the quality assessment of the analyzed web application. In fact, its overall quality could be judged satisfying in case the tool identifies a large number of false clones using the no-single cluster configuration. On the other hand, the quality of a web application could be considered inadequate when the number of false clones identified by using the no-single cluster configuration is low.

	SEKE 02	**SRA**	**EC**
Trade-off configuration	39	12	19
No-single cluster configuration	11	5	22

Table 13. Trade-off and break-even configurations of k-means on the analyzed web applications

6.4 Remarks on LSI

The analysis of the results achieved at the content level enabled us to provide some considerations on the used information retrieval technique. In particular, we observed that better results were obtained in case a stemming algorithm was used to reduce inflected (or sometimes derived) words to their stem. In particular, we observed that on SEKE 02 (i.e., the web applications with English content) the considered clustering algorithms produced better results in terms of precision and recall values.

Stemming algorithms on the content of the static pages of EC and of SRA (i.e., the web applications presenting information in Italian language) were not used. This was due to the lack of an appropriate stemming algorithm for the Italian language. Furthermore, we also observed that better values of precision and recall were achieved when neither stop word list nor stop word functions were employed. Such a result could be due to the fact that the static text contained in the pages was not much. However, to confirm or to contradict these results a further investigation is required on larger web application.

Finally, Table 14 shows the dimension of the k space achieved by employing the Guttman-Kaiser on the web site and the web applications used within the case study. We can observe a small reduction of the concept space as compared to the number of analyzed pages. This could be due to the low number of static pages and to the low number of different concepts. It is easy to imagine that on web applications containing more concepts a sensible reduction of the concept space should be obtained.

	SEKE 02	SRA	EC
Dimension of the reduced space	47	21	152
Largest possible dimension of the concept space	49	24	160

Table 14. Number of singular values computed using the Guttman-Kaiser criterion

7 Remarks and future work

In this paper we have investigated the effect of using different clustering algorithms in grouping similar pages at the structural and content level. In particular, we have taken into account three variants of the agglomerative hierarchical clustering algorithm [369], a divisive clustering algorithm [32], a variant of k-means [24] partitional clustering algorithm, and a competitive clustering algorithm, namely Winner Takes All (WTA) [22]. Pages are compared at the structural level applying the Levenshtein algorithm [36] to the strings that encode the sequences of HTML tags, while Latent Semantic Indexing is used to get pages dissimilarity at the content level.

The usefulness of identifying pages that are similar or cloned at the structural level is commonly recognized as evidenced by the different clustering based approaches proposed in the past [12, 13, 17, 43, 45]. On the other hand, approaches aimed at identifying page similarity at the content level have been marginally investigated [44]. Nevertheless, these approaches could be successfully employed to reverse engineer and reengineer web applications. For example, the navigational schema of a given existing web application could be enhanced adding links between pairs of pages that are considered similar at the content level. The new navigational schema should provide a better support to browse and search information among the pages of the considered application. Moreover, groups of pages similar at the content level could be also employed to identify the application domain entities of a web based system and the relation among them [2]. Static pages containing nearly identical content could also be reengineered in dynamic pages maintaining the similar content in a separate file or in a database. Furthermore, as we only consider static text our clustering based process might be applied in application domains different from the maintenance of web based systems. For example, it could be useful to group search results of web or desktop search engines. The clustering of search results is not new and has been widely investigated in the information retrieval field [47].

The selected clustering algorithms have been compared on the static pages of two web applications and on one static web site. The analysis of the results achieved on these applications revealed that at the structural and content level the clustering algorithms produced similar values of precision and recall. However, at the structural level the WTA clustering algorithm suggests some guidelines to support the software engineer in pruning bad configurations and quickly identifying the best one. In particular, we observed that more than one configuration enables the identification of a number of clusters equal to the number of expected ones. Indeed, selecting as configurations numbers ranging from the number of web pages to 1, we noted that after a given configuration the number of identified and expected clusters always coincides (i.e., the break-even configuration). We experimentally observed that the trade-off configurations have been obtained around the break-even configuration. To confirm or contradict this result a further investigation on larger and different web applications is however required.

At the content level guidelines to support the pruning of the worse configurations have been identified as well. In fact, we observed that using the k-means in most cases the no-single cluster configuration (i.e., the larger number of expected clusters such that the k-means identifies non single clusters) generally produced an F-measure value similar to the one obtained by using the trade-off configuration. In case different F-measure values were obtained we noted that the no-single cluster configuration produced a better or equal recall value. This means that a higher number of false clones are identified. However, less effort is generally required to remove false clones with respect to identify pages that are true clones especially when the web application contains a large number of pages.

One of the contributions of this paper is the definition of heuristics to reduce the effort to identify pages that are similar at the structural and content. However, some considerations regarding the use of the investigated clustering algorithms are due. Indeed, we observed that at the content level we generally obtained worse results. This could be due to the adopted dissimilarity measures. To address this issue a further investigation is required. For example, we could adapt and improve information retrieval techniques that have been commonly adopted in the past to implement web and desktop search engines [9, 47]. Future work will be also devoted to investigate the effect of using different characteristic of the page structure (e.g., the attribute of the tag of a given page). A further direction would be to consider the navigation structure of a given web application, thus decomposing it into groups of functionally related components and improving the result quality of a clustering based approach. We also plan to combine different dissimilarity measures to improve the accuracy of the groups of pages that are similar at the content and structural level. The use of different clustering algorithms will be also taken into consideration as future work.

Acknowledgements

The authors would like to thank Angelo Ciaramella and Rocco Oliveto for the stimulating discussion and the precious suggestions. Special thanks are also due to Francesco Casertano, who implemented some software components of the system prototype. The work described in this paper is supported by the project METAMORPHOS (MEthods and Tools for migrAting software systeMs towards web and service Oriented aRchitectures: exPerimental evaluation, usability, and technology tranSfer), funded by MiUR (Ministero dell’Università e della Ricerca) under grant PRIN-2006-2006098097. The work described in this paper is also supported by the project TOCAI (Tecnologie Orientate alla Conoscenza per Aggregazioni di Impresa in internet) under grant FIRB 2005.

References

1. Anquetil N. and Lethbridge T. C., Experiments with clustering as a software remodularization method. In Proc. of 6th Working Conference on Reverse Engineering, 1999, pp.235-255.
2. Antoniol G., Canfora G., Casazza G., and De Lucia A., Web Site Reengineering using RMM. In Proc. of International Workshop on Web Site Evolution, 2000, pp. 9-16.
3. Baker B. S., On finding duplication and near duplication in large software systems. In Proc. of the 2nd Working Conference on Reverse Engineering, 1995, pp. 86-95.
4. Balazinska M., Merlo E., Dangenais M., Lague B., and Kontogiannis K., Measuring Clone Based Reengineering Opportunities. In Proc. of 6th IEEE International Symposium on Software Metrics, 1999, pp. 292-303.
5. Baxter D., Yahin A., Moura L., Sant’Anna M., and Bier L., Clone Detection Using Abstract Syntax Trees. In Proc. of IEEE Intl Conference on Software Maintenance, 1998, pp. 368-377.

6. Boldyreff C., Munro M., and Warren P. The evolution of websites. In Proc. of 7th IEEE International Workshop on Program Comprehension, 1999, pp. 178-185.
7. Boldyreff C. and Kewish R., Reverse Engineering to Achieve Maintainable WWW Sites. In Proc. of 8th IEEE Working Conference on Reverse Engineering, Suttgart, 2001, pp. 249-257.
8. Boldyreff C. and Tonella P., Special Issue of J of Software Maintenance, vol.16, no.1-2, 2004.
9. Brin S. and Page L. The Anatomy of a Large-Scale Hypertextual Web Search Engine. In Computer Networks and ISDN Systems, vol. 30, no. 1-7, 1998, pp. 107-117.
10. Calefato F., Lanubile F., and Mallardo T., Function Clone Detection in Web Applications: A Semiautomated Approach. In Journal of Web Engineering, vol. 3, no. 1, 2004, pp. 3-21.
11. Conallen J., Building Web application with UML, 2000.
12. De Lucia A., Francese R., Scanniello G., and Tortora G., Identifying Cloned Navigational Patterns in Web Applications. In Journal of Web Engineering, vol. 5, no. 2, 2006, pp. 150-174.
13. De Lucia A., Scanniello G., Tortora G., Identifying Similar Pages in Web Applications using a Competitive Clustering Algorithm. In Intl J on Software Maintenance and Evolution, vol. 19, no. 5, pp. 281-296.
14. De Lucia A., Risi M., Scanniello G., and Tortora G., Clustering Algorithms and Latent Semantic Indexing to Identify Similar Pages in Web Applications In Proceedings of 9th IEEE International Symposium on Web Site Evolution, 2007, pp. 65-72.
15. De Lucia A., Fasano F., Oliveto R., and Tortora G., Recovering traceability links in software artifact management systems using information retrieval methods. In Transaction on Software Engineering and Methodology, vol. 16, no. 13, 2007.
16. Deerwester S., Dumais S. T., Furnas G. W., Landauer T. K., and Harshman R., Indexing by Latent Semantic Analysis. In J of American Society for Information Science, no.41, 1990, pp.391-407.
17. Di Lucca G. A., Di Penta M., and Fasolino A. R., An Approach to Identify Duplicated Web Pages. In Proc. of 26th Annual Intel Computer Software and Application Conference, 2002, pp. 481-486.
18. Di Lucca G. A., Fasolino A. R., De Carlini U., Pace F., and Tramontana P., Comprehending web applications by a clustering based approach. In Proc. Of 10th Intl Workshop on Program Comprehension, 2002, pp. 261-270.
19. Di Lucca G. A., Fasolino A. R., Taralli F., and De Carlini U., Testing Web Application. In Proc. of International Conference on Software Maintenance, 2002, pp. 310-319.
20. Di Lucca G. A., Fasolino A. R., De Carlini U., and Tramontana P., Abstracting Business Level UML Diagrams from Web Applications. In Proc. of 5th IEEE International Workshop on Web Site Evolution, 2003, pp.12-19.
21. Di Lucca G. A., Fasolino A. R., and Tramontana P., Reverse engineering Web applications: the WARE approach. In J of Software Maintenance and Evolution: Research and Practice, vol.16, no. 1-2, 2004, pp. 71-101.
22. Duda R. O., Hart P. E., and Stork D. G., Pattern Classification. In Wiley-Interscience Publication, pp. 576-581.
23. Eichmann D., Evolving an Engineered Web. In Proc. of International Workshop Web Site Evolution, 1999, pp. 12-16.
24. Flynn P.J., Jain A. K., and Murty M. N., Data Clustering: A Review, In ACM Computing Surveys, vol. 31, no. 3, 1999, pp. 264-323.
25. Ginige A. and Murugesan S., Special issue on Web Engineering. IEEE Multimedia, vol. 8, no. 1-2, 2001.
26. Guttman L., Some necessary conditions for common factor analysis. Psychometrika, vol. 19, 1954, pp. 149-61.
27. Harman D., Ranking Algorithms. In Information Retrieval: Data Structures and Algorithms, 1992, pp. 363-392.
28. Higo Y., Ueda T., Kamiya Y., Kusumoto S., and Inoue K., On software maintenance process improvement based on code clone analysis. In Proc. of 4th Intl Conference on Product Focused Software Process Improvement, 2002, pp. 185-197.

29. Isakowitz T., Stohr E. A., and Balasubramanian P. RMM: a Methodology for Structured Hypermedia Design. In Communications of the ACM, vol. 38, no. 8, 1995, pp. 34-44.
30. Jain K. and Dubes R. C., Algorithms for clustering data. Prentice-Hall Advanced Reference Series, 1988.
31. Kaiser H. F., The application of electronic computers to factor analysis. Educational and Psychological Measurement, vol. 20, 1960, pp. 141-51.
32. Kaufman L. and Rousseeuw P.J., Finding Groups in Data. An Introduction to Cluster Analysis. Wiley, New York, 1990.
33. King F., Step-wise clustering procedures. In Journal of the American Statistical Association, vol. 62, 1967, pp. 86-101.
34. Kohonen T., Self-organizing formation of topologically correct feature maps. In Biological Cybernetics, vol. 43, 1982, pp. 59-69.
35. Kuhn A., Ducasse S., and Girba T. Semantic Clustering: Identifying Topics in Source Code. In International Journal of Information and Software Technology, vol. 43, no 3, 2007, pp. 230-243.
36. Levenshtein V. L.: Binary codes capable of correcting deletions, insertions, and reversals. In Cybernetics and Control Theory, vol. 10, (1966), pp. 707-710.
37. Maletic, J. I. and Marcus, A. Supporting program comprehension using semantic and structural information. In Proc. of 23rd Intl Conference on Software Engineering, 2001, pp. 103-112.
38. Mcqueen J., Some methods for classification and analysis of multivariate observations. In Proc of 5th Berkeley Symposium on Mathematical Statistics and Probability, 1967, pp. 281-297.
39. Nakov. P., Latent semantic analysis for german literature investigation. In Proceedings of the International Conference, 7th Fuzzy Days on Computational Intelligence, Theory and Applications, 2001, pp. 834-841.
40. Oudshoff A. M., Bosloper I.E., Klos T. B., and Spaanenburg L., Knowledge discovery in virtual community texts: Clustering virtual communities. In Journal of Intelligent and Fuzzy Systems, vol. 14, no. 1, 2003, pp. 13-24.
41. Rajapakse C. and Jarzabek S., An Investigation of Cloning in Web Applications. In Proc. of 5th International Conference on Web Engineering, 2005, pp. 252-262.
42. Ricca F. and Tonella P., Understanding and Restructuring Web Sites with ReWeb. In IEEE Multimedia, vol. 8, no. 2, 2001, pp. 40-51.
43. Ricca F. and Tonella P., Using Clustering to Support the Migration from Static to Dynamic Web Pages. In Proc. of International Workshop on Program Comprehension, 2003, pp. 207-216.
44. Ricca F., Tonella P., Girardi C., and Pianta E., Improving Web site understanding with keyword-based clustering, J of Software Maintenance and Evolution Research and Practice, vol.20, no.1, 2008, pp. 1–29.
45. Tonella P., Ricca F., Pianta E., and Girardi C., Restructuring Multilingual Web Sites. In Proc. of International Conference on Software Maintenance, 2002, pp. 290-299.
46. Tonella P., Ricca F., Pianta E., Girardi C., Di Lucca G., Fasolino A. R., and Tramontana P., Evaluation methods for Web application clustering. In proc. of 5th IEEE Symposium on Web Site Evolution, 2003, pp. 33- 40.
47. Van Rijsbergen C. J., Information Retrieval, second ed., Butterworth, London, 1979.
48. Wiggerts T. A., Using clustering algorithms in legacy systems remodularization. In Proc. of 4th Working Conference on Reverse Engineering, 1997, pp. 33-43.