Interpreting Clustering Results through Cluster Labeling

Onaiza Maqbool¹, Haroon A. Babri²,
Lahore University of Management Sciences¹
Lahore, Pakistan
University of Management and Technology²
Lahore, Pakistan
onaiza@lums.edu.pk, babri@umt.edu.pk

Abstract

Software architecture refers to the overall structure of a software system, and is defined by the components (sub-systems) within a software system and their interactions with one another. Quite often, there is little documentation describing a software system’s architecture, especially in the case of legacy software systems. Thus techniques must be employed for recovering the architecture from the software’s source code. Given the size and complexity of legacy systems, researchers have started exploring the use of automated techniques for architecture recovery. A technique that has shown promising results is clustering. Clusters that are obtained as a result of the clustering process represent sub-systems within a software system, but are not easy to interpret until they are given appropriate names. In this paper, we present a cluster labeling scheme based on identifiers. As the clustering process proceeds, keywords are ranked using the inverse document frequency ranking scheme. Results of experiments conducted on a test system demonstrate that our labeling approach is effective. We also compare the clustering results of the complete algorithm and the weighted combined algorithm based on labels of the clusters produced by them during clustering.

Index Terms — Architecture recovery, Clustering, Cluster labeling, Weighted combined algorithm

1. Introduction

Software architecture refers to the overall structure of a software system, and is defined by the components (sub-systems) within a software system and their interactions with one another. The specification of overall system structure becomes a significant issue as the size and complexity of software systems increase [1]. A clear understanding of the architecture is required to allow adaptation of a system to changing requirements, and also to reduce costs by sharing components across several projects [2]. Although software systems have had architectures ever since the first program was divided into modules, these architectures have mostly been implicit with little documentation describing them [1].

The problem is further aggravated in the case of legacy software systems, which are software systems that support the day-to-day operations of a business, and have been in use for a number of years. According to Lehman, these systems must have undergone changes in their lifetime, in order to remain useful [3]. In the case of legacy systems, even if architectural documentation was developed for the original system, it is unlikely that the documentation was updated to reflect changes made to it over the years.

In the absence of up-to-date documentation describing the architecture of software legacy systems, it is useful to employ techniques for recovering their architecture from the source code. Given the size and complexity of these systems, manual techniques are infeasible. Thus there has been growing interest in exploring automated techniques for recovering software architectures.

A technique that has shown promising results is cluster analysis. Clustering has been employed in many domains including biology, astronomy, psychology, to form group of items possessing similar characteristics. In recent years, there has been growing interest in the application of clustering techniques for automating the architecture recovery of legacy systems [4], [5], for re-
modularization [6], [7], [8], and also for detection of objects in structured code [9].

Clusters that are obtained as a result of the clustering process need to be labeled appropriately in order to understand the purpose of each cluster and to evaluate the effectiveness of clustering. Clusters may be labeled manually, but it is obviously more useful to obtain labels automatically. It has been pointed out by researchers that automatic cluster labeling has not received much attention, despite the fact that it is an important issue [10]. Schwanke [11] briefly describes a technique to label clusters based on how many times a feature is used in a cluster. By utilizing this information, and also drawing on knowledge of the code, short titles are manually selected for the clusters. Tzerpos [10] uses a pattern based approach to recognizing and labeling clusters. The identified patterns are expected to occur in large systems with around 100 source files. Tonella [12] describes the use of keywords within web pages to cluster and label similar pages.

In this paper, we present a scheme for labeling clusters which utilizes identifiers to represent an entity. To rank keywords associated with entities as clustering proceeds, we use inverse document frequency. We perform clustering using two clustering algorithms, the complete algorithm and the weighted combined algorithm. The labels obtained are used to evaluate the effectiveness of the labeling scheme and also of the clustering process. We also compare the clustering results of the two algorithms by comparing the labels obtained as a result of their application.

The organization of this paper is as follows. In section 2 we present an overview of clustering techniques. Section 3 details our approach to clustering and cluster labeling. Section 4 presents experimental results of applying our labeling scheme to a test system. Finally, we present the conclusions.

2. An overview of clustering

The clustering process groups together entities based on the similarity of their features. The purpose is to form groups of entities such that entities within a group are similar to each other and different from entities in other groups. For software clustering, commonly employed entities are functions or source files. Features, which represent characteristics of an entity, are often binary in the case of software. A ‘1’ denotes the presence of a feature and a ‘0’ indicates its absence. Formal features are based on the code e.g. variables or types used by an entity, whereas informal features may be based on comments or identifiers within the code. Table 1 represents a hypothetical software system with 3 entities and 4 features.

<table>
<thead>
<tr>
<th>Entities</th>
<th>Feature1</th>
<th>Feature2</th>
<th>Feature3</th>
<th>Feature4</th>
</tr>
</thead>
<tbody>
<tr>
<td>E₁</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>E₂</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>E₃</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1: Entities and associated feature vectors

The next step in the clustering process is to identify similar entities. Commonly employed measures for determining similarity in other domains include distance measures e.g. the Euclidean distance, or similarity coefficients e.g. the Correlation coefficient. However, in the case of software, if the feature vectors are binary, association coefficients, which are appropriate for binary features, may be employed. A listing of popular association coefficients is available in [13]. For software, experimental results indicate that the Jaccard association coefficient gives the best results [7], [8]. The Jaccard coefficient $J$ is given by:

$$J = a/(a+b+c)$$

where $a$ represents the number of features which are present in both entities, and $b$, $c$ represent features which are present in one entity and absent in the other. A similarity table is constructed to represent the similarity between every pair of entities within the system. The similarity table for the system presented in Table 1 is depicted in Table 2.

<table>
<thead>
<tr>
<th>Entities</th>
<th>$E₁$</th>
<th>$E₂$</th>
<th>$E₃$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E₁$</td>
<td>2/3</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td>$E₂$</td>
<td></td>
<td></td>
<td>1/4</td>
</tr>
</tbody>
</table>

Table 2: Similarity table (Jaccard coefficient)

After the determination of similarities, clustering algorithms are employed to form clusters of similar entities. A taxonomy of clustering algorithms can be found in [14]. Agglomerative hierarchical clustering algorithms, which proceed in a bottom-up manner to form clusters, are suitable for software clustering since they allow clusters (software sub-systems) to be viewed at different abstraction levels. At every step, these algorithms cluster the two most similar entities, eventually forming a single cluster containing all entities. Thus the clusters formed are hierarchical, with earlier iterations representing a detailed view of the system, and later iterations representing a high-level view of the system’s architecture. We may select an appropriate cutoff point to view clusters at the required level of abstraction.

As clusters are formed, similarity between clusters needs to be determined. Different agglomerative clustering algorithms address the issue of determining clustering similarity in various ways. Some well known hierarchical algorithms are [13]:

### References

[6], [7], [8], [9], [10], [11], [12], [13], [14]
- Single Linkage
  \[ \text{sim}(A, \text{BUC}) = \max(\text{sim}(A, B), \text{sim}(A, C)) \]
- Complete Linkage
  \[ \text{sim}(A, \text{BUC}) = \min(\text{sim}(A, B), \text{sim}(A, C)) \]
- Weighted Average Linkage:
  \[ \text{sim}(A, \text{BUC}) = \frac{1}{2}(\text{sim}(A, B) + \text{sim}(A, C)) \]
- Unweighted Average linkage:
  \[ \text{sim}(A, \text{BUC}) = \frac{(\text{sim}(A, B) \cdot \text{size}(B) + \text{sim}(A, C) \cdot \text{size}(C))}{\text{size}(B) + \text{size}(C)} \]

where \( A, B \) and \( C \) represent entities and \( \text{BUC} \) represents the cluster formed by merging \( B \) and \( C \). Experimental results indicate that the complete linkage algorithm gives the best results for software clustering in terms of the most cohesive clusters [7], [8].

In [15] and [16], we proposed the combined and weighted combined algorithms for software clustering. When two entities \( A \) and \( B \) are clustered to form \( C \), the combined algorithm associates a new feature vector with \( C \) by taking the binary OR of the feature vectors of \( A \) and \( B \). Similarity between the various entities and the newly formed cluster is then re-calculated. The weighted combined algorithm, which is a refinement of the combined algorithm, forms the new feature vector for \( C \) by taking the sum of features in the constituent entities \( A \) and \( B \), and normalizes this sum through division by the cluster size. Since the feature vector no longer remains binary in this case, we use the non-binary counterpart of the Jaccard coefficient, i.e. the Ellenberg measure, for calculating similarity between entities. The Ellenberg measure \( E \) is given by:

\[
E = \frac{1/2 \cdot Ma}{1/2 \cdot Ma + Mb + Mc}
\]

where \( Ma \) represents the sum of features present in both the entities and \( Mb, Mc \) represent the sum of features present in one entity and not in the other.

The Ellenberg measure often favors merging clusters rather than entities, resulting in large, sometimes non-cohesive clusters. To discourage formation of such non-cohesive clusters, we proposed the Unbiased Ellenberg measure in [16]. The Unbiased Ellenberg measure \( E_u \) is given by:

\[
E_u = \frac{1/2 \cdot Ma}{1/2 \cdot Ma + b + c}
\]

where \( Ma \) represents the sum of features present in both the entities and \( b, c \) represent the number of features present in one entity and not in the other.

A popular means of evaluating clustering results is to compare them with a decomposition of the software system prepared manually by human experts. Such a method of evaluation is known as an expert decomposition or gold standard. External assessment was carried out for evaluation in [16], and our experimental results indicated that the combined and weighted combined algorithms together with the Jaccard and Unbiased Ellenberg measures respectively, present better clustering results for software as compared to the complete algorithm.

In this paper, we use cluster labeling to make it easier to understand the purpose of each cluster formed during clustering. In order to evaluate the effectiveness of our labeling scheme, we compare the automatically assigned labels with labels assigned by a human expert. We also use the labels obtained to compare the clustering results of the complete algorithm and the weighted combined algorithm.

### 3. Our labeling approach

#### 3.1. The clustering process

Labels were assigned to clusters at each step during the clustering process. Clustering consisted of the following steps:

- **Entity and feature selection**
  Functions were selected as entities to be clustered. The following features were used:
  - Global variables used by an entity
  - User defined types used by an entity
  - Calls made by an entity

- **Selection of similarity measures**
  The Jaccard and Unbiased Ellenberg measures were used as similarity measures, since it has been demonstrated [15], [16] that they give good results for software clustering.

- **Selection of clustering algorithm**
  The complete algorithm and weighted combined algorithm were used to perform clustering.

- **Selection of evaluation method**
  We carried out external assessment of cluster labeling results by comparison with labels obtained manually by a human expert to evaluate the effectiveness of our labeling scheme. Moreover, we compared clustering results of the complete and weighted combined algorithms by comparing labels of the clusters produced by them.

#### 3.2. The labeling process

The following approach was adopted for cluster labeling:
- Label selection
  Since we are using functions as entities, we used function identifiers to represent a cluster. In case function identifiers consist of composite words, they are parsed to obtain individual words e.g. init_ellipse_drawing, consists of three keywords init, ellipse and drawing.

- Label ranking
  As entities are combined to form clusters, the keywords associated with each entity must be combined and given an appropriate weight (ranked) to form the cluster label. A simple approach is to use frequency to rank keywords e.g. when two entities, init_spline and init_line are clustered, the init keyword is ranked highest since it has a frequency of 2 as compared to spline and line which have frequencies of 1 each.
  Another approach is that of inverse document frequency [12] which is defined as:
  \[ I[n] = F[n] * \ln(D/d[n]) \]
  where \( F[n] \) represents the number of occurrences of the \( n \)th keyword, \( d[n] \) represents the number of clusters containing it and \( D \) is the total number of clusters. The inverse document frequency approach gives less weight to a keyword if it is present in other entities/clusters within the system e.g. if the init keyword is present in 9 other clusters in a system consisting of 20 clusters, its weight would reduce to 1.39 in comparison to spline and line, whose weight would increase to 2.99 if they are not present in any other clusters within the system. The justification for this approach is that a commonly occurring keyword represents a domain concept rather than characteristics of a certain cluster, hence it is less useful for distinguishing a cluster from other entities and clusters.
  We employed the inverse document frequency approach to rank the keywords.

4. Experiments and results

4.1 The test system

The software we have chosen for our cluster labeling experiments is Xfig version 3.2.3, which is an open source drawing utility that runs under the X Window System. It has been written in C, and consists of around 75,000 lines of code. The design documentation of Xfig is not available, although the user manual and other useful information is available at the Xfig site [17].

The Xfig system consists of five major sub-systems, whose source code files can be identified by their names. Some relevant statistics of these sub-systems are provided in Table 3:

<table>
<thead>
<tr>
<th>System</th>
<th>Purpose</th>
<th>Files</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>d_*files</td>
<td>Drawing tasks</td>
<td>10</td>
<td>94</td>
</tr>
<tr>
<td>e_*files</td>
<td>Editing tasks</td>
<td>19</td>
<td>369</td>
</tr>
<tr>
<td>f_*files</td>
<td>File related tasks</td>
<td>15</td>
<td>139</td>
</tr>
<tr>
<td>u_*files</td>
<td>Utility files</td>
<td>18</td>
<td>422</td>
</tr>
<tr>
<td>w_*files</td>
<td>Window related tasks</td>
<td>30</td>
<td>637</td>
</tr>
</tbody>
</table>

In this paper we present clustering results for the d_*files subsystem only. The d_files subsystem consists of 94 functions. Moreover, the Xfig system makes use of 1746 global variables and 828 user defined types.

The source files for the Xfig system have been parsed using the Rigi tool and relevant ‘facts’, which represent entities and their features have been stored in an exchange format called the ‘Rigi Standard Format (RSF)’ [18]. Facts of interest to us include:
- Global variables accessed by a function
- User defined types accessed by a function
- Functions called by a function.

4.2 Analysis of results

In this section, we present the labeling results when clustering is applied to the d_*_files subsystem of Xfig and analyze these results. We make two kinds of comparisons:
- Comparison between automatic labeling and labeling by human experts
- Comparison between the clustering results of the complete and weighted combined algorithms based on labeling

Comparison was performed between the automatically assigned labels and labels assigned by human experts in order to evaluate the effectiveness of our labeling scheme. To perform the comparison, results of the clustering process were viewed at an abstraction level close to the expert’s view of the software system. Results of the comparison are presented in Table 4:
Table 4: Comparison between automatic labels and manually assigned labels

<table>
<thead>
<tr>
<th>Weighted Combined</th>
<th>Human Expert</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inverse Frequency</td>
<td>Human Expert</td>
</tr>
<tr>
<td>Init</td>
<td>Initialize functions</td>
</tr>
<tr>
<td>Selected</td>
<td>Select functions</td>
</tr>
<tr>
<td>Create</td>
<td>Create functions</td>
</tr>
<tr>
<td>Cancel</td>
<td>Cancel functions</td>
</tr>
<tr>
<td>Text</td>
<td>Text manipulation functions</td>
</tr>
<tr>
<td>Char/Input</td>
<td>Character manipulation functions</td>
</tr>
<tr>
<td>Prefix/suffix</td>
<td>Prefix/postfix manipulations</td>
</tr>
<tr>
<td>Sub spline</td>
<td>Sub spline functions</td>
</tr>
<tr>
<td>Blinking cursor</td>
<td>Cursor functions</td>
</tr>
<tr>
<td>-</td>
<td>Getpoints functions</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Complete</th>
<th>Human Expert</th>
</tr>
</thead>
<tbody>
<tr>
<td>Init/drawing</td>
<td>Initialize functions</td>
</tr>
<tr>
<td>Selected</td>
<td>Select functions</td>
</tr>
<tr>
<td>Create</td>
<td>Create functions</td>
</tr>
<tr>
<td>Cancel</td>
<td>Cancel functions</td>
</tr>
<tr>
<td>Text</td>
<td>Text manipulation functions</td>
</tr>
<tr>
<td>Do</td>
<td>Char manipulation functions</td>
</tr>
<tr>
<td>Prefix/suffix</td>
<td>Prefix/postfix manipulations</td>
</tr>
<tr>
<td>Sub spline</td>
<td>Sub spline functions</td>
</tr>
<tr>
<td>Blinking</td>
<td>Cursor functions</td>
</tr>
<tr>
<td>-</td>
<td>Getpoints functions</td>
</tr>
</tbody>
</table>

The following observations can be made:
- The human expert decomposes the d_*files into 10 sub-systems. Based on the clustering and labeling results, it is clear that most of the sub-systems identified by human experts have also been identified by clustering.
- There is correspondence between automatically assigned labels and labels assigned by human experts. Thus a labeling scheme based on function identifiers and inverse document frequency is meaningful and may be used to summarize the purpose of each cluster.

A comparison between the labels assigned by the complete and weighted combined algorithms yields interesting results. It was observed that when forming clusters of small sizes, the two algorithms agree to a high degree. This is apparent from the similar names assigned to such clusters by both algorithms. Difference was observed in the results when larger clusters are merged. Figure 1 and Figure 2 represent a high-level view of the software system, as taken by the two algorithms towards the end of the clustering process:

- The following observation can be made:
  - The weighted combined algorithm identifies **drawing** and **text** to be the two major sub-systems of the d_*files system. The complete algorithm labels the two major sub-systems as **selected** and **create**.
  - The labels assigned by the weighted combined algorithm are meaningful and easy to interpret. The assigned labels reveal that the d_*files system consists of **drawing** functions and **text** functions. A study of the d_*files source code reveals that this view is correct.
  - The labels assigned by the complete algorithm are less meaningful as compared to those assigned by the weighted combined algorithm. The assigned labels identify **selected** functions and **create** functions to be the two major sub-systems of the d_*files system. The identified sub-systems are not easily understandable.

From the labeling results presented in this section, we can conclude that clusters produced by the complete and weighted combined algorithms are similar during the earlier stages of the clustering process. However, as clustering proceeds, the weighted combined algorithm forms more meaningful clusters towards the later stages. This is clear from the labels assigned to the clusters during the clustering process.

5. Conclusions

In this paper, we described clustering as a technique for architecture recovery of legacy systems. Clusters obtained as a result of the clustering process need to be labeled to ease interpretation of clustering results. In this paper, we presented a labeling scheme based on function
identifiers as representative keywords of an entity. To rank these keywords as clusters are formed, we employed inverse document frequency.

We performed clustering experiments on the d_*files system of Xfig version 3.2.3 using the complete and weighted combined clustering algorithms. Labels were automatically assigned using the labeling scheme described above. Experimental results reveal that our labeling scheme is effective and representative of an expert’s view of the software system. Moreover, a comparison of the clusters produced by the weighted combined and complete algorithms, by using the assigned labels, reveals that the clusters produced by both the algorithms are similar during the initial phases of the clustering process. However, during the later phases, clusters formed by the weighted combined algorithm are more meaningful, as is clear from the labels assigned to larger clusters.

In the future we intend to perform similar experiments on other software systems. We would also like to use other ranking schemes, so that we may compare their results and identify the labeling scheme which is most effective in the software context.

References