Abstract

As the age of software systems increases they tend to deviate from their actual design and architecture. It becomes more and more difficult to manage and maintain such systems. We explore the idea of software clustering for reverse engineering and re-modularization. Clustering together software artifacts provides an automatic technique for discovering high level abstract entities within a system. Previous work on software clustering has identified many areas where further investigation is required. Clustering techniques should be tuned to the type of system they are being applied to. In this paper we explore a new clustering algorithm called the ‘combined’ algorithm which, as our experiments show, provides more promising results for software clustering than the previously used algorithms. We also analyze the behavior of correlation and distance metrics for binary features.

Index Terms—Software Clustering, Combined algorithm, Correlation, Binary Features

1. Introduction

The field of cluster analysis is not new. It has been applied in many disciplines to discover similarities between artifacts or to find a taxonomy of similar objects. It has also been extensively applied in computer science to discover patterns within data. The application of cluster analysis to software artifacts is comparatively a newer area of research where lots of issues have not been addressed. The major goal of software clustering is to automate the process of discovering high level abstract subsystems within the source code. Such subsystems can be used for software visualization, re-modularization and architecture discovery. The higher level subsystems also aid the maintenance engineers to gain a better understanding of source code and make the necessary changes. Lung points out that software clustering can be applied to software during various life cycles [1]. It can support software architecture partitioning during the forward engineering process and aid in recovering software architecture during the reverse engineering processes.

The process of clustering together similar objects depends upon the nature and size of data. It also depends upon the similarity measures and clustering algorithms being used. Clustering itself can impose a structure on the data it is being applied to. When applying a clustering technique one has to keep in mind the nature of the data and use the appropriate parameters suited to that type of data.

When clustering together software entities with a point of view of obtaining better re-modularization we would aim for high cohesion and low coupling [2]. For example if functions that access the same variables are placed into the same module, the module would tend to have communicational cohesion.

The above argument brings out the need to tailor our clustering algorithm to the type of software being clustered. Researchers have pointed out that different clustering techniques behave differently when applied to different types of software systems [2], [3]. A technique might impose an artificial structure on the existing system instead of bringing out the natural one. In this paper we would like to introduce a new clustering algorithm called the ‘combined’ algorithm which, according to our experiments, has shown better performance than other algorithms when grouping together software entities. It is appropriate in the case of binary feature vectors where the presence and the absence of a feature is taken into account for grouping objects or entities together.

The organization of the paper is as follows. In section 2 we present an overview of software clustering techniques. Section 2 also presents an analysis of the
correlation metric and distance metrics for binary features and details the combined algorithm. Section 3 gives the results of applying clustering techniques to an open source drawing utility called Xfig. Finally, we discuss our results and present the conclusions.

2. Clustering approach

Wiggerts [4] gives an overview of software clustering techniques and suggests the use of the term ‘entity’ to describe elements being grouped together and ‘features’ to denote the attributes of these entities. Generally, for software clustering, files, functions or procedures of a program are considered as entities. References to variables by a function or calls to other modules by a function or its naming convention are treated as the features of that entity [2], [3]. A possible clustering generated by a clustering technique is called a partition. The following issues need to be addressed when considering software clustering processes:

- The entities that should be grouped together
- The similarity measures that can be used to group together similar entities
- The clustering algorithm to be applied
- Evaluation of a partition

2.1. Features and entities

Similar to the approach followed by Davey and Burd [2], we also use functions as entities. Clustering of functions is easier to understand and represents the functionality of the system more clearly than files. Files may be used if the test system is large and contains a very large number of functions, otherwise the use of functions is appropriate. Our test system (described in section 3) contains around 1700 functions only, making the function a more suitable choice for an entity.

We associate the following types of features with each entity:

- Global: Global variables referred to by the entity
- Type: The data type or user defined type accessed by the entity
- Call: The functions called by the entity

The reason for using global and type features is that they reflect cohesion present between the entities, something we aim for when clustering software entities. The call feature represents functional bindings. [2] states that features based on data provide better results than the call feature.

A feature vector is associated with each entity in the system. The features are binary features with a one denoting the presence of a feature and a zero denoting the absence. Since an entity represents a function declared in the program, the features accessed by that entity are very small and its feature vector is usually sparse.

In this paper, we take into account only the presence of features and do not explore the effect of the nature of a feature on similarity e.g. we do not differentiate between whether a global variable is simply being ‘set’ or is being ‘used’ by an entity. By examining a feature in more detail, we may obtain a more refined re-modularization.

2.2. Similarity metrics

The metrics used to calculate the similarity between two objects are the association coefficients, correlation measures and distance metrics. Similarity is the inverse of distance and the greater the distance between two objects, the less similar they are. Correlation also gives an estimate of similarity. The higher the correlation the more similar the two entities are. Some of the similarity / distance metrics used in software clustering are [4]:

- Sorensen-Dice (association coefficient)
- Jaccard metric (association coefficient)
- Simple metric (association coefficient)
- Euclidean distance (distance measure)
- Camberra metric (distance measure)
- Pearson product moment correlation

To calculate the association coefficients between two entities E1 and E2 a table of similarities is constructed as given by [4]:

\[
\begin{array}{c|cc}
E2 & & \\
\hline
1 & a & b \\
\hline
0 & c & d \\
\end{array}
\]

In the above table ‘a’ represents the count of features present in both E1 and E2. ‘b’ represents the total number of features present in E1 but absent in E2 and so on. The following association coefficients are then defined as:

- The Jaccard co-efficient is given by:
  \[ J = \frac{a}{(a+b+c)} \]
- The Simple co-efficient
  \[ S = \frac{a+d}{(a+b+c+d)} \]
- The Sorensen-Dice co-efficient is
  \[ SD = \frac{2a}{2a+b+c} \]

We can see that the Sorensen-Dice and Jaccard metrics are based on the presence of features, whereas the simple coefficient also takes into account the absence of features. The values of all the association coefficients lie between 0 and 1 with 1 denoting maximum similarity. So far none of the researchers has tried to relate the association coefficients with the correlation and distance metrics. The correlation coefficient between X and Y is given by:


\[
\rho = \frac{\sum_{i=1}^{n} (x_i - \bar{x}_i)(y_i - \bar{y}_i)}{\sqrt{\sum_{i=1}^{n} (x_i - \bar{x}_i)^2 \sum_{i=1}^{n} (y_i - \bar{y}_i)^2}}
\]

For binary features the correlation metric reduces to [10]:

\[
\rho = \frac{4ad - bc}{(a+b+c)(a+b+d)}
\]

If the value of d is much greater than a, b or c then this metric reduces to the following form:

\[
\rho = \frac{a}{(a+b)(a+c)} \quad \text{for} \ d \gg a, \ d \gg b \text{ and } d \gg c
\]

We can see the above form resembles the Jaccard metric.

The Euclidean distance between X and Y is given by:

\[
D(X,Y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}
\]

For binary features the Euclidean distance reduces to [10]:

\[
D = \sqrt{b+c}
\]

Also, the Camberra distance metric C is given by:

\[
C(X,Y) = \sum_{i=1}^{n} \frac{|x_i - y_i|}{|x_i + y_i|}
\]

If the above term is taken as zero for a case when both \(x_i\) and \(y_i\) are zero then Camberra metric reduces to [10]:

\[
C = b+c
\]

The Camberra metric and the Jaccard metric are related as:

\[
J = \frac{a}{(a+C)}
\]

It is relevant to note that clustering using similarity metrics is applied across a wide variety of domains. A similarity measure may be well suited to one domain but not to another e.g. some similarity measures give weight to the absence of a feature. Such a similarity measure may be appropriate in applications where the absence of a particular feature in two entities is a sign of similarity between the two. For software clustering, this does not seem to be relevant for the features we have used. e.g. the fact that two entities do not access a global variable or a data type does not indicate that they are cohesive.

We make the following observations from the above discussion and our experiments (Section 3 and [10]):

- The Jaccard and Sorensen-Dice metrics perform almost identically. Figure 1 illustrates this result. The result has also been pointed out by [2]. This indicates that assigning more weight to the features present in both the entities does not have a significant impact on the results.
- The simple metric does not seem to be appropriate for dealing with software clustering as it takes into account the absence of features.
- The correlation metric is equivalent to Jaccard metric when the total absent features are very large as compared to the count of present features or mismatched features. This can be seen from Figure 1. Since a function will normally access very few of the total global variables, types and functions within the whole system, it is usually the case that total absent features are very large.
- For clustering, the Camberra distance metric and Euclidean distance will perform equivalently.
- Similarity will be low if \(b\) and \(c\) are high, Jaccard will be low and both Camberra and Euclidean distances (inverse of similarity) would be high. However, when dealing with software the Camberra metric and Euclidean distance are not appropriate for a case when all of \(a\), \(b\) and \(c\) are zero. In this case the distance is zero indicating high similarity which is misleading. For example two functions like sort and swap may not be accessing any common features so that the distance between them would be zero. The distance metrics, in this case, indicate high similarity which does not seem intuitively correct.

Because of the above reasons we will present our results with the Jaccard metric which seems to be the most intuitive for software entities.

### 2.3. Clustering methods and the combined algorithm

For clustering we chose the agglomerative hierarchical algorithm which involves the following steps:

- Form a raw data matrix of binary feature values. The rows denote entities and the columns denote the individual features.
- From the data matrix calculate the similarity matrix using one of the similarity metrics described previously. The similarity values are calculated for each individual pair of entities.
- Find two entities with maximum similarity. Merge the two entities together into a new cluster and calculate the new cluster distance from other entities using any one of 4 linkage algorithms (described below). Repeat merging the entities till only one cluster remains.

The various clustering algorithms that define how to calculate the distance or similarity between the new cluster and the remaining entities are described below: \((A,B,C)\) are singleton clusters, \(\text{sim}(A,B)\) indicates similarity between A and B, \(\text{BUC}\) is the newly merged cluster.

- **Single Linkage**

  \[
  \text{sim}(A, \text{BUC}) = \text{Max}(\text{sim}(A,B), \text{sim}(A,C))
  \]

- **Complete Linkage**


sim(A,BUC)=Min(sim(A,B),sim(A,C))

- **Weighted Average Linkage:**
  sim(A,BUC)=1/2(sim(A,B))+1/2(sim(A,C))

- **Unweighted Average Linkage:**
  sim(A,BUC)=(sim(A,B)*size(B)+(sim(A,C)*size(C))/(size(B)+size(C))

Experimental results indicate that the performance of complete linkage is better than single linkage [2],[3]. Also, the performance of weighted and unweighted average linkage algorithms lies somewhere in between the single and complete linkage algorithms.

Since, we are clustering together software entities which have binary feature vectors associated with them, it seems more intuitive to associate a new feature vector with the newly formed cluster. The new feature vector is calculated by taking the OR between the two vectors. For example if entity A accesses a variable G1 and entity B accesses the global variable G2 then the cluster, formed by merging A and B, accesses both G1 and G2. We call this new algorithm the combined algorithm. An example would clarify how the combined algorithm works. Suppose there are three entities in the system given by A,B,C whose feature vectors are given by:

<table>
<thead>
<tr>
<th>Entity</th>
<th>Feature Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>{1 1 0 0 0 1}</td>
</tr>
<tr>
<td>B</td>
<td>{0 1 1 0 0 1}</td>
</tr>
<tr>
<td>C</td>
<td>{0 0 0 1 1}</td>
</tr>
</tbody>
</table>

The corresponding similarity table using the Jaccard coefficient is given by:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>-</td>
<td>1/2</td>
<td>1/3</td>
</tr>
<tr>
<td>B</td>
<td>1/2</td>
<td>-</td>
<td>1/3</td>
</tr>
<tr>
<td>C</td>
<td>1/3</td>
<td>1/3</td>
<td>-</td>
</tr>
</tbody>
</table>

After the first iteration A and B are merged together into cluster AB and the new data matrix is given by:

<table>
<thead>
<tr>
<th>Entity</th>
<th>Feature Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB</td>
<td>{1 1 1 0 0 1}</td>
</tr>
<tr>
<td>C</td>
<td>{1 0 0 0 0 0}</td>
</tr>
</tbody>
</table>

The four entities are such that D has one feature in common with A,B and C. The corresponding similarity table using the Jaccard coefficient is given by:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>-</td>
<td>1/3</td>
<td>0</td>
<td>1/4</td>
</tr>
<tr>
<td>B</td>
<td>1/3</td>
<td>-</td>
<td>0</td>
<td>1/4</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>1/4</td>
</tr>
<tr>
<td>D</td>
<td>1/4</td>
<td>1/4</td>
<td>1/4</td>
<td>-</td>
</tr>
</tbody>
</table>

After the first iteration A and B are merged together into cluster AB. D has now 2 features in common with the cluster AB and only one in common with C. Thus it is more suitable that D be clustered with AB in the next iteration rather than with C. This is what the combined algorithm would do. On the other hand, if the complete algorithm in used, it will indicate D to be equivalently similar with both AB and C. In this case, D may be clustered with C.

### 2.4. Evaluation of results

The process of clustering is a sophisticated one, and as such no definite quantitative measures exist to evaluate the quality of a clustering partition. Davey and Burd [2] and Anquetil and Lethbridge [3] have used precision and recall as measures to evaluate the performance of the clustering algorithm. For this purpose, the test clustering is compared with a reference system also called the expert decomposition or expert partition. It is best that the designer or the developer of the system makes the expert partition as they have the best knowledge of the system under consideration. Precision and recall are defined using the concept of **intra pairs**. Intra pairs are the pairs of entities in the same cluster. Precision and recall are defined by taking all the intra pairs in the expert decomposition and the test clustering and are given by:

- **Precision** : Percentage of intra pairs in the test clustering that are also in the expert decomposition.
- **Recall** : Percentage of intra pairs in the expert decomposition that are also in the test clustering.

Mathematically, if A denotes the set of intra pairs of the test clustering and B represents the set of intra pairs of the expert partition then the precision $p$ and recall $r$ are given by:

$$p = \frac{|A \cap B|}{|A|}, \quad r = \frac{|A \cap B|}{|B|}$$

Recall measures the percentage of intra pairs that are relevant to the expert clustering. Ideally a recall of 100% would be required. Precision, on the other hand measures
From the above definitions it is clear that there is a tradeoff between precision and recall. It is desirable that both precision and recall are high for a test clustering. However, this is generally not the case. When using agglomerative clustering, at the start of the clustering process there are all singleton clusters and the system has zero recall and 100% precision. As the clustering proceeds the precision decreases and the recall increases. When the entire system is one big cluster the recall is 100% but the precision is very low.

3. Experiments and results

3.1. The test system

We used Xfig Version 3.2.3 as a test system for conducting experiments with clustering software entities. Xfig is an open source drawing tool that runs under X-Windows system [7]. Xfig was also used as a model system for carrying out a competition of the reverse engineering tools [8]. It is written in the C programming language and consists of 75K lines of source code. Its source code is distributed over about a 100 source files and around 75 include files. There is no documentation regarding the structure or implementation of Xfig. However, usage manuals are available for this system.

Xfig source code has been parsed using the Rigi parser [5] by the Rigi Group. The facts are in Rigi Standard Format (RSF) which is a Tuple Attribute (TA) format [6]. The RSF file has around 200,000 facts including the facts for calls, accesses, references etc. Analysis of Xfig shows around 1700 functions. The RSF file was used for software clustering. It was exported to an SQL compliant RDBMS to ease the access of facts during the clustering process. In Xfig a consistent naming convention of source code files is used as follows:

- `d_*` files are intended for drawing shapes
- `e_*` files are related to editing
- `f_*` files have file related functions
- `u_*` files pertain to utilities for drawing and editing
- `w_*` files contain the X-windows related calls

The above files can be used to classify the entire system into sub-systems. We conducted experiments with the above subsystems to evaluate the combined algorithm.

To assess the results we used precision and recall with respect to an expert decomposition. To calculate the precision and recall of a test partition, an expert decomposition is required. We constructed the expert decomposition by placing all the functions in one source file in one cluster. This is based on the assumption that all functions placed in one file perform similar functions.

3.2. Comparison of complete and combined algorithm

Since the complete linkage algorithm gives the best partitions out of single, weighted, unweighted linkage algorithms, we compare its performance with the combined algorithm [2],[3]. To conduct this comparison we plotted the precision and recall graphs for the Jaccard similarity measure when using all features in Figure 2 for the d-files sub-system of Xfig. Davey and Burd [2] use the crossover point as a criterion to evaluate the results of clustering. The crossover point for precision and recall signifies the point when the total number of intra pairs of the expert clustering is equal to the total number of intra pairs in the test clustering. When precision is greater than recall, it means that the total number of intra pairs in the algorithm’s test clustering is less than that of the expert’s. Conversely, when the total number of pairs in the expert clustering is less than that of the test clustering then the recall is higher than precision. The graphs of Figure 2 illustrate this tradeoff between precision and recall (the iterations till the end are not shown).

As can be seen from Figure 2 the crossover point for precision and recall for the combined algorithm is higher than for the complete algorithm. Figure 2 presents the height of the crossover point for different sub-systems of Xfig when using the Jaccard similarity metric and all features. For the d-files, e-files and f-files sub-systems the performance of the combined algorithm is much better than that of the complete linkage algorithm. Also, it is equivalent for the w-files subsystem and a little worse for the u-files subsystem.

<table>
<thead>
<tr>
<th></th>
<th>Complete</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>d_files</td>
<td>18%</td>
<td>33%</td>
</tr>
<tr>
<td>e_files</td>
<td>22%</td>
<td>29%</td>
</tr>
<tr>
<td>f_files</td>
<td>25%</td>
<td>30%</td>
</tr>
<tr>
<td>u_files</td>
<td>33%</td>
<td>30%</td>
</tr>
<tr>
<td>w_files</td>
<td>17%</td>
<td>17%</td>
</tr>
</tbody>
</table>

4. Conclusions

In this paper we explored the notion of software clustering for extracting high level subsystems to be used in software maintenance, software re-modularization, program visualization etc. We analyzed the distance and correlation metric for binary features and showed that the behavior of the correlation metric is similar to Jaccard metric when the number of absent features is very large as compared to the present features. We also defined a new
clustering algorithm called the ‘combined’ algorithm. We compared this algorithm with the complete linkage algorithm and found its performance to be better than the complete linkage algorithm on the Xfig system.

Further work is required to evaluate the performance of the combined algorithm on other systems. It may be particularly useful to conduct experiments on systems for which design documentation is available. Further, we are investigating the effect of the nature of features on clustering. Assigning weights to features may provide better clusters, which we would like to examine in more detail.

References


Figure 1: Comparison of similarity metrics for the d-files subsystem using all features and complete algorithm. Jaccard and Sorenson-Dice perform equivalently. Results of Correlation and Jaccard are very similar.

Figure 2: Precision and recall for the complete and combined algorithms for the d_files subsystem when using the Jaccard similarity measure on all features

Acknowledgment

Thanks to Johannes Martin of the Rigi group, University of Victoria, Canada for his help and providing us with the RSF for Xfig. We also acknowledge Dr.Humayum M. Mian, Chairman PITB, for his help and support during the initial phases of this work.