INTERNATIONAL JOURNAL OF RESEARCH IN COMPUTER APPLICATIONS AND ROBOTICS
ISSN 2320-7345

COOPERATIVE CLUSTERING USING OPTIMIZED SIMILARITY MEASURES

Diya Davis
Post-Graduate Student
Department of Computer Science and Engineering,
Karunya University
diyadavis2011@gmail.com
India

Bright Gee Varghese
Assistant professor
Department of Computer Science and Engineering,
Karunya University
brightachus@karunya.edu
India

***Abstract* - Clustering is a useful technique to group related data entities. Many different algorithms have been proposed for software clustering. To combine the strengths of various algorithms, Consensus Based Techniques (CBTs) can be used, where more than one actors (e.g. algorithms) cooperate together to achieve a common goal. The Cooperative Clustering Technique (CCT), a type of CBT, which employs cooperation among more than one similarity measures during the hierarchical clustering process. Cooperative Clustering is capable of showing significant improvement over individual clustering algorithms for software modularization.**

***Keywords:* Software clustering process, Feature vector cases, Cooperative software clustering**

I. INTRODUCTION

A well-documented architecture can improve the quality and maintainability of a software system. However, many existing systems often do not have their architecture documented. Maintenance of architectural documentation is one of many problems that confront today's large software projects. Software clustering holds out the promise of helping in this task. During the maintenance of a software system, most of the effort is usually devoted to understanding the structure of the software system. This task is facilitated if a system is well modularized with less coupling and maximum cohesion, making it easier to change it and also to evaluate the side effects of a change. Coupling is the degree of dependency between the modules and Cohesion is the inter-dependency within a single module. Low coupling is the sign of a well-structured software system and a good design. When combined with high cohesion, it provides high reliability and maintainability. The important application of cluster analysis is to modularize a software system by grouping together software entities that are similar or related to each other and thereby achieve minimum coupling and maximum cohesion.

Software clustering is the process of decomposing a software system into meaningful subsystems. It is the process of finding similar groups of entities in data. Entities within a cluster have similar characteristics or features, and are dissimilar from entities in other clusters. Clustering thus provides a high-level view of the system by grouping together related or similar software entities. Software clustering plays an important role in understanding legacy software systems, assisting in their architectural documentation, and supporting their re-modularization [1]. A number of clustering algorithms and measures have been proposed and applied for software modularization. To improve clustering results, recently proposed Consensus Based Techniques (CBTs) suggest that multiple clustering techniques be combined. Traditionally, CBTs have been studied as the integration of more than one clustering algorithm in a single process to achieve a common goal [6].

The Consensus Based Cooperative Clustering Technique (CCT) employs cooperation between algorithms at intermediate steps during the clustering process. This technique can be implemented for clustering algorithms which are iterative in nature [6], [7]. CCT allows parallel execution of algorithms, which support each other by exchanging information at each iteration. The intermediate-level cooperation suggested in CCT need not be restricted to cooperation between algorithms only; a broader view may be taken. For example, cooperation may be in the form of more than one similarity measure producing results at each iteration, which are combined during the clustering process. Cooperative techniques have been explored in many disciplines. Although the idea of cooperation between techniques for software has not been applied for modularization, CCT produces better modularization results and it is capable of showing significant improvement over individual clustering algorithms for software modularization [1] - [4].

II. AN OVERVIEW OF SOFTWARE CLUSTERING PROCESS

A. *Selection of Entities and Features*

Selection of entities and features differs across various software systems. For structured software systems, entities can be files or functions in the system. Features can be global variables or user defined types used by an entity [8]. For object oriented software systems, entities can be classes, and features can be the relationships between classes such as inheritance or containment. Features can be classified into two types: binary or non-binary [9]. Binary features simply indicate the presence or absence of a relationship [10]. Non-binary features are capable of indicating the strength of a relationship and may be scored on different scales [11]. Features are usually binary in the software domain. A software system must be parsed to extract entities and features before applying a clustering algorithm. The result of this extraction is an (n × p) matrix, where 'n' is the number of entities and 'p' is the number of features. Table 1 presents an (n × p) matrix containing 4 entities (E1–E4) and 6 binary features (f1–f6).

Table 1
An example of an (n × p) feature matrix

	f1	**f2**	**f3**	**f4**	**f5**	**f6**
E1	0	1	1	0	1	0
E2	1	1	1	0	0	1
E3	1	0	0	1	0	1
E4	1	0	0	1	1	0

B. *Selection of Similarity Measures*

In the second step, a similarity measure is applied to compute similarity between every pair of entities, resulting in a similarity matrix. Selection of a similarity measure should be done carefully, because selecting an appropriate similarity measure may influence clustering results more than the selection of a clustering algorithm [5].

1) *Similarity Measures for Binary Features:*

To determine the similarity between two entities, different similarity measures may be used. Table 2 lists some well-known similarity measures for binary features.

Table 2
Similarity measures for binary features

Name	**Mathematical representation**
Jaccard	a/(a+b+c)
Russell & Rao	a/(a+b+c+d)
Simple Matching	(a+d)/(a+b+c+d)
Sorensen Dice	2a/(2a+b+c)
Sokal Sneeth	a/(a+2(b+c))
Rogers-Tanimoto	(a+d)/(a+2(b+c)+d)

For two entities Ei and Ej, a is the number of features that are present, i.e.,'1' in both entities Ei and Ej, b represents the number of features that are present in Ei but absent in Ej, c represents the number of features that are not present in Ei and are present in Ej, and d represents the number of features that are absent, i.e., '0' in both entities. n = a + b + c + d is the total number of features [1], [2].

2) *Similarity Measures for Non-binary Features:*

Table 3 lists some well-known similarity measures for non-binary features. In Table 3, Ma represents the sum of features that are present in both entities Ei and Ej, Mb represents the sum of features that are present in Ei but are absent in Ej and Mc represents the sum of features that are not present in Ei and are present in Ej.

For software modularization, it has been shown that the Jaccard measure produces better results than other measures for binary features [8], [12]. One reason for this is that it does not consider d (absence of feature/negative match) [13], [14]. For non-binary features, a counterpart of the Jaccard similarity measure called the 'Unbiased Ellenberg' produces better results for software clustering.

Table 3
Similarity measures for non-binary features

Name	**Mathematical representation**
Ellenberg	0.5*Ma/(0.5* Ma+Mb+Mc)
Unbiased Ellenberg	0.5*Ma/(0.5*Ma+b+c)
Gleason measure	Ma/(Ma + Mb + Mc)
Unbiased Gleason	Ma/(Ma + b + c)

III. FEATURE VECTOR CASES

The Cooperative Clustering technique involves the cooperation of more than one similarity measure during the clustering process. Thus to apply the cooperative technique in software clustering for modularization, the first step is to analyze similarity measures to identify their strengths and weaknesses. For this purpose, it selected similarity measures that have been widely used for software modularization and have shown better results than other measures, i.e., the Jaccard measure (for binary features) and the Unbiased Ellenberg measure (for non-binary features) [10]. Some preliminary work [2], [3] identified certain cases where these well known similarity measures may produce poor results. For these cases, we proposed new measures for binary as well as for non-binary features. The identified cases and the proposed similarity measures are presented in this section.

A. *Feature Vector Case 1 (fvc1)*

An example feature matrix with four entities (E1–E4) and four features (f1–f4) for this case is presented in Table 4. From Table 4, it can be seen that for the entities E1 and E2, a = 2, and for E3 and E4, a = 4. The corresponding similarity matrix using the Jaccard measure is given in Table 5.

Table 4
Example A

Entities	**f1**	**f2**	**f3**	**f4**
E1	1	1	0	0
E2	1	1	0	0
E3	1	1	1	1
E4	1	1	1	1

Table 5
Similarity matrix using Jaccard for Example A

Entities	**E1**	**E2**	**E3**	**E4**
E1	-			
E2	1	-		
E3	0.5	0.5	-	
E4	0.5	0.5	1	-

It can be seen from Table 4 that the Jaccard measure finds entities E1 and E2, E3 and E4 to be equally similar although they have different values of a. Thus any algorithm which uses the Jaccard measure for calculating similarity will take an arbitrary decision.

B. *Feature Vector Case 2 (fvc2)*

An example feature matrix for fvc2with four entities (E1–E4) and six features (f1–f6) is presented in Table 6. It can be seen thatE1 and E2 are completely similar whereas E3 and E4 share a greater number of common features as compared to E1 and E2 but are not completely similar.

Table 6
Example B

Entities	**f1**	**f2**	**f3**	**f4**	**f5**	**f6**
E1	1	1	0	0	0	0
E2	1	1	0	0	0	0
E3	1	1	1	1	1	0
E4	1	1	1	1	0	1

Table 7
Similarity matrix using Jaccard for Example B

Entities	**E1**	**E2**	**E3**	**E4**
E1	-			
E2	1	-		
E3	0.4	0.4	-	
E4	0.4	0.4	0.6	-

The corresponding similarity matrix for Table 6 using the Jaccard measure is given in Table 7. It can be seen from Table 7 that the Jaccard measure finds entities E1 and E2 to be most similar, since they share two features f1 and f2. On the other hand, E3 and E4 are found to be less similar although they share four features, because of feature f5 which is accessed by E3 but not by E4 and feature f6which is accessed by E4 but not by E3.The feature vectors of E3 andE4 indicate more common functionality. In this case, it may be more useful to cluster the entities sharing a larger number of features (greater value of a) even if there are a few b's and c's indicating differences.

From Table 6 for fvc2 it can be seen that E3 and E4 have twice the number of common features as compared to E1 and E2, i.e., if a1represents the number of common features for E1 and E2, and a2represents the common features associated with E3 and E4, thenE3 and E4 have a2 = 2 × a1. Value of b2 + c2 ≤ a1 as can be seen in Table 6.

C. *Feature Vector Case 3 (fvc3)*

Consider Table 8 having four entities (E1–E4) and five features (f1–f5). All the entities have the same value of a = 3 but entities E1and E2 have b and c = 0 while E3 and E4 have b = 1 and c = 1. The corresponding similarity matrix using Russell & Rao is given in Table 9. It can be seen from Table 9 that Russell & Rao considers all entities to be equally similar.

This behavior of Russell & Rao may lead to a large number of arbitrary decisions. In this case, it may be more useful to cluster the entities sharing common features with b = c = 0 [1] – [3].

Table 8
Example C

Entities	**f1**	**f2**	**f3**	**f4**	**f5**
E1	1	1	1	0	0
E2	1	1	1	0	0
E3	1	1	1	1	0
E4	1	1	1	0	1

Table 9
Similarity matrix by Russell & Rao for Example C

Entities	**E1**	**E2**	**E3**	**E4**
E1	-			
E2	0.6	-		
E3	0.6	0.6	-	
E4	0.6	0.6	0.6	-

D. *Russel&Rao-NM – a new similarity measure for binary features*

From the above discussion, it is possible to conclude that problems arise in the Jaccard measure because it does not consider the proportion of common features as compared to the total features, and the problem in Russell & Rao is because it depends only on a whatever the values of b, c may be. To solve these problems, a new Russell & Rao-like similarity measure is introduced which overcomes the deficiencies of the Jaccard measure as well as the Russell & Rao measure. It does this by retaining the proportion of common features as compared to the total features through n = a + b + c + d in the denominator (similar to Russell & Rao), but overcomes deficiency in the Russell &Rao measure by adding a + b + c to the denominator (similar to Jaccard) thus not depending on a only. Our new measure is defined as follows:

$$\text{Russell\&Rao -NM} = \frac{a}{a+b+c+d+n} \qquad (1)$$

where *n* is the total number of features.

$$\text{Russell\&Rao -NM} = \frac{a}{2(a+b+c+d)} \qquad (2)$$

The application of the Russell & Rao-NM measure to fvc1, fvc2 shows that it overcomes the deficiencies of the Jaccard measure for these cases, and its application to fvc3shows that it overcomes the deficiency of the Russell & Rao measure. The number of iterations to obtain an efficient clustering can be effectively reduced by the new Russell & Rao-NM similarity measure.

IV. COOPERATIVE CLUSTERING USING RUSSELL&RAO -NM

The main goal of analyzing the (n × p) matrix is to determine strengths and weaknesses of the measures, thus allowing us to incorporate more than one measure in a single process. Therefore, for feature vector cases, where one measure performs better it can be used, and for the other cases another measure can be used to calculate the similarity between pairs of entities. In this way, it performs cooperative clustering by integrating more than one measure for different cases in a single clustering process. Thus in our case the definition of cooperative clustering is "where more than one similarity/distance measures cooperate in a clustering process". The new cooperative clustering approach and algorithm is described in the following section.

A. *Definitions and Notations*

Before describing the algorithms, we present some definitions and notations used.
Definition 1: PESyM is a Pair of Entities in the Symmetric Similarity Matrix. If we have two entities Ei and Ej then (Ei, Ej) = (Ej, Ei) and i /= j. Entity (Ei or/and Ej) may also represent a cluster containing more than one entity.
Definition 2: The PESyM-MaxSim is the Pair of Entities in the Symmetric Similarity Matrix Against Maximum Similarity value. PESyM-MaxSim is denoted by M.

We also use the following notations. VS1maxand VS2maxare the maximum values in similarity matrix S1and S2, respectively. c* represents a new cluster. RRsim is the Russell & Rao similarity measure and RRNMsim is the Russell & Rao-NM similarity measure. NC is the Number of Clusters and LC is a Large Cluster.

B. *Cooperative OUSM Software Clustering*

The main steps in the cooperative software clustering technique are given in Algorithm 5.1. The Cooperative OUSM (COUSM) algorithm first creates a similarity matrix S1for the PESyM belonging to fvc1and fvc2, using the Russell & Rao-NM measure. If no PESyM belong to fvc1or fvc2, it assigns a '0' similarity value. Then the similarity matrix S2is created for all the PESyM in the given (n x p) matrix using the Russell & Rao measure. The algorithm then searches for the maximum similarity value in the similarity matrix S1to make a cluster. If the similarity value in S1is equal to '0' then the algorithm searches for the maximum inS2. During this process, if the maximum is from similarity matrixS1, then the algorithm will update both similarity matrices S1and S2 (Step 20). If the maximum value is from S2, then the algorithm will update only similarity matrix S2 (Step 22).

C. *Algorithm Using Russel&Rao-NM*

```
Input: (n × p) matrix.
Output: Hierarchy of Clusters
1:  for $i$ = 1 to $(n-1)$ do
2:    for $j$ = 2 to $n$ do
3:      if $(E_i, E_j) \in fvc_3$ then
4:         $S_1(i,j) = RRNM_{sim}(E_i, E_j)$
5:      else
6:         $S_1(i,j) = 0$
7:      end if
8:      $S_2(i,j) = RR_{sim}(E_i, E_j)$
9:    end for
10: end for
11: $bool$ = Notfinished
12: repeat
13:   if (($V^{s1}_{max}$ > 0) & ($bool$ /= Finished)) then
14:      $c* = Mv^{s1}_{max}$
15:   else
16:      $c* = Mv^{s2}_{max}$
17:      $bool$ = Finished
18:   end if
19:   if ($bool \neq$ Finished) then
20:      Update $S_1$and $S_2$, using basic linkage
algorithm between $PESyM$ of $c^*$ and $(E_n - c^*)$
21:   else
22:      Update $S_2$, using basic linkage algorithm
between $PESyM$ of $c^*$ and $(E_n - c^*)$
23:   end if
24: until Required $NC$ or one $LC$ is formed
```

V. CONCLUSION

In the software domain, an important application of cluster analysis is to modularize a software system by grouping together similar or related software entities. A number of clustering algorithms have been employed for software modularization. More than one algorithm can be combined to improve the strengths of various algorithms. The Cooperative Clustering Technique allows clustering algorithms to cooperate at intermediate steps during the clustering process. Till now, no work has been done to apply CCT to modularize software systems. This paper suggests that CCT can be applied for software modularization to improve results. A single similarity measure may not appropriately handle the various situations that arise during the software clustering process. Cooperation between similarity measures is therefore a promising approach to significantly improve results as compared to the individual clustering algorithms for software modularization.

VI. REFERENCES

[1] Rashid Naseem, Onaiza Maqbool, Siraj Muhammad., Cooperative clustering for software modularization. The journal of Systems and Software., 2013.

[2] Naseem, R., Maqbool, O., Muhammad, S. Improved similarity measures for software clustering. In: Proceedings of the European., 2011.

[3] Naseem, R., Maqbool, O., Muhammad, S., An improved similarity measure for binary features in software clustering. In: Proceedings of the International Conference on Computational Intelligence, Modelling and Simulation, pp. 111–116., 2010.

[4] Choi, S., Cha, S., Tappert, C. A survey of binary similarity and distance measures. Journal of Systemics, Cybernetics and Informatics 8 (1), 43–48., 2010.

[5] Wen, Z., Tzerpos, V., Evaluating similarity measures for software decompositions. In: Proceedings of 20th IEEE International Conference on Software Maintenance, pp. 368–377., 2004.

[6]Kashef, R., Kamel, M.S. Cooperative clustering. Journal of Pattern Recognition43 (6)., 2010.

[7] Mitra, S., Banka, H., Pedrycz, W. Collaborative rough clustering. Pattern Recognition and Machine Intelligence, 768–773., 2005.

[8]Anquetil, N., Lethbridge, T. Experiments with clustering as a software remodularization method. In: Proceedings of Sixth Working Conference on Reverse Engineering, pp. 235–255., 1999.

[9]Abbasi, A.Q. Application of appropriate machine learning techniques for automatic modularization of software systems. Mphil. thesis, Quaid-i-Azam University Islamabad., 2008.

[10]Maqbool, O., Babri, H.A. Hierarchical clustering for software architecture recovery. IEEE Transactions on Software Engineering 33 (11), 759–780., 2007.

[11]Wiggerts, A. Using clustering algorithms in legacy systems remodularization. In: Proceedings of the 4th Working Conference on Reverse Engineering, pp. 33–43., 1997.

[12] Davey, J., Burd, E. Evaluating the suitability of data clustering for software re-modularisation. In: Proceedings of Working Conference on Reverse Engineering, pp. 268–276., 2000.

[13]Maqbool, O., Babri, H.A. The weighted combined algorithm: a linkage algorithm for software clustering. In: Proceedings of the European Conference on Software Maintenance and Reengineering, pp. 15–24., 2004.

[14] Saeed, M., Maqbool, O., Babri, H.A., Hassan, S., Sarwar, S. Software clustering techniques and the use of combined algorithm. In: Proceedings of the European Conference on Software Maintenance and Reengineering, pp. 301–306., 2003.