Cooperative clustering for software modularization

Rashid Naseem a, Onaiza Maqbool b,*, Siraj Muhammad c

a Department of Computer Science, City University of Science and Information Technology, Peshawar, Pakistan
b Department of Computer Science, Quaid-i-Azam University, Islamabad, Pakistan
c Department of Computer Science, Shaheed Benazir Bhutto University Sheringal, Dir (U), Pakistan

ARTICLE INFO

Article history:
Received 9 February 2012
Received in revised form 21 January 2013
Accepted 14 March 2013
Available online xxx

Keywords:
Cooperative clustering
Software clustering
Jaccard-NM
Unbiased Ellenberg-NM
Feature vector cases

ABSTRACT

Clustering is a useful technique to group data entities. Many different algorithms have been proposed for software clustering. To combine the strengths of various algorithms, researchers have suggested the use of Consensus Based Techniques (CBTs), where more than one actors (e.g. algorithms) work together to achieve a common goal. Although the use of CBTs has been explored in various disciplines, no work has been done for modularizing software. In this paper, the main research question we investigate is whether the Cooperative Clustering Technique (CCT), a type of CBT, can improve software modularization results. The main contributions of this paper are as follows. First, we propose our CCT in which more than one similarity measure cooperates during the hierarchical clustering process. To this end, we present an analysis of well-known measures. Second, we present a cooperative clustering approach for two types of well-known agglomerative hierarchical software clustering algorithms, for binary as well as non-binary features. Third, to evaluate our proposed CCT, we conduct modularization experiments on five software systems. Our analysis identifies certain cases that reveal weaknesses of the individual similarity measures. The experimental results support our hypothesis that these weaknesses may be overcome by using more than one measure, as our CCT produces better modularization results for test systems in which these cases occur. We conclude that CCTs are capable of showing significant improvement over individual clustering algorithms for software modularization.

© 2013 Elsevier Inc. All rights reserved.

1. Introduction

Clustering 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. In the software domain, an important application of clustering is to modularize a software system by grouping together related or similar software entities, thus providing a high-level view of the system. 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 (Forestier et al., 2010; Kashef and Kamel, 2010). Traditionally, CBTs have been studied as the integration of more than one clustering algorithm in a single process to achieve a common goal.

The Consensus Based Cooperative Clustering Technique (CCT) employs cooperation between algorithms at intermediate steps during the clustering process (Kashef and Kamel, 2010; Mitra et al., 2005). This technique can be implemented for clustering algorithms which are iterative in nature, e.g., hierarchical algorithms which have \((n - 1)\) iterations \((n\) is the number of entities to be clustered). 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 disciplines, e.g., bioinformatics and for text documents (Kashef, 2008). Although the idea of cooperation between techniques for software clustering was presented in Maqbool and Babri (2007), it has not been applied for modularization.

In preliminary work in Naseem et al. (2010), we presented two situations (cases) which highlighted deficiencies in the Jaccard similarity measure for binary features. The Jaccard measure has been widely applied for software modularization and has shown better results than other measures. We then proposed the Jaccard-NM measure which overcomes the identified deficiencies. Results of the two measures were compared by applying clustering to modularize three proprietary software systems written in C++. In Naseem et al. (2011), we extended our work by considering non-binary features and introduced the Unbiased Ellenberg-NM measure. Moreover, we compared our proposed Jaccard-NM measure with the Russell &
Rao measure (Choi et al., 2010), and presented a case which highlighted a deficiency in the Russell & Rao measure. Experiments were carried out to modularize three proprietary software systems written in C++. The Jaccard, Jaccard-NM and Russell & Rao measures were applied individually during the clustering process to evaluate which measure produced better results by comparing the automatically produced decomposition with an expert decomposition produced by a human expert.

In this paper, the main research question we investigate is whether the Cooperative Clustering Technique can be applied for software modularization to improve results. This question was not investigated in earlier papers. In this context, we investigate the following additional research questions. (1) Can more than one similarity measure be made to cooperate during the clustering process? (2) Is there a relation between arbitrary decisions taken during the clustering process and the modularization results? (3) Does the cooperative technique improve results for different types of hierarchical clustering algorithms for binary as well as non-binary features?

The contributions of this paper may be summarized as:

1. We use cooperative clustering for modularizing software systems, which has not been done earlier. Moreover, the Cooperative Clustering Technique, as defined in Kashef and Kamel (2010) and Mitra et al. (2005), employs cooperation among clustering algorithms at intermediate steps during the clustering process, whereas we define it to be the cooperation of more than one similarity measure during clustering. To employ more than one similarity measure, we present an analysis of well-known measures and identify certain situations which reveal their weaknesses. We then present our similarity measure which can cooperate with existing measures in these situations to overcome the weaknesses.

2. We present an approach for cooperative clustering for two types of well known software clustering algorithms, (a) widely used agglomerative hierarchical algorithms which only update the similarity matrix (OUSM) at each step of the clustering process, e.g., Complete Linkage, Single Linkage (Anquetil and Lethbridge, 1999), and (b) more recently proposed new feature vector (NFV) algorithms which make a new feature vector of the newly formed cluster during the clustering process, e.g., Weighted Combined algorithm (Maqbool and Babri, 2004) and LIMBO (Andritos and Tzerpos, 2005). Our approach works for binary as well as for non-binary features.

3. We carry out modularization experiments on five software systems and compare the results of cooperative clustering with clustering using individual similarity measures. Quantitative evaluation is carried out by comparing automatically produced clustering results with decompositions prepared manually by human experts. To provide insight into the results, the number of arbitrary decisions taken by each measure during the clustering process is also presented.

The rest of the paper is organized as follows. In Section 2, we describe steps in clustering. Section 3 presents feature vector cases to highlight strengths and weaknesses of well known similarity measures for software clustering. In Section 4, the cooperative clustering approach is described in detail. Section 5 presents the experimental setup. In Section 6, the experimental results for the cooperative algorithms are discussed. Section 7 describes the threats to validity. Section 8 presents the related work. Finally in Section 9, we give the conclusions.

2. An overview of the software clustering process

In this section, we provide an overview of the steps in the clustering process.

2.1. Selection of entities and features

Selection of entities and features varies across software systems. For structured software systems, files or functions are usually selected as entities. Features may be, for example, global variables or user defined types used by an entity (Anquetil and Lethbridge, 1999). For object oriented software systems, entities may be classes (Abassi, 2008), and features are typically defined by the relationships between classes, e.g., inheritance or containment. Features may be binary or non-binary. Binary features simply indicate the presence or absence of a relationship (Maqbool and Babri, 2007). On the other hand, non-binary features are capable of indicating the strength of a relationship and may be scored on different scales e.g. ordinal, interval (Wiggerts, 1997). In the software domain, features are usually binary.

Before applying a clustering algorithm, a software system must be parsed to extract entities and features. The result is an \((n \times p)\) matrix, where ‘\(n\)’ is the number of entities and ‘\(p\)’ is the number of features. Each entity in the \((n \times p)\) matrix has a feature vector \(f_i\), where \(f_i = [f_{i1}, f_{i2}, f_{i3}, \ldots, f_{ip}]\). Table 1 presents an \((n \times p)\) matrix containing 4 entities \((E1\text{--}E4)\) and 6 binary features \((f1\text{--}f6)\).

2.2. 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 (Wen and Tzerpos, 2004b).

2.2.1. 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.

In Table 2, \(a, b, c\) and \(d\) can be determined using Table 3. For two entities \(E_i\) and \(E_j\), \(a\) is the number of features that are present, i.e., ‘1’ in both entities \(E_i\) and \(E_j\), \(b\) represents the number of features that are present in \(E_i\) but absent in \(E_j\), \(c\) represents the number of features that are not present in \(E_i\) and are present in \(E_j\), 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.

<table>
<thead>
<tr>
<th>S. no.</th>
<th>Name</th>
<th>Mathematical representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Jaccard</td>
<td>[a/(a+b+c)]</td>
</tr>
<tr>
<td>2</td>
<td>Russell &amp; Rao</td>
<td>[a/(a+b+c+d)]</td>
</tr>
<tr>
<td>3</td>
<td>Simple Matching</td>
<td>[(a+d)/(a+b+c+d)]</td>
</tr>
<tr>
<td>4</td>
<td>Sorensen Dice</td>
<td>[2a/(2a+b+c)]</td>
</tr>
<tr>
<td>5</td>
<td>Sokal Sneath</td>
<td>[a/(a+2b+c)]</td>
</tr>
<tr>
<td>6</td>
<td>Rogers-Tanimoto</td>
<td>[(a+d)/(a+2b+c+d)]</td>
</tr>
</tbody>
</table>
Table 3
Contingency table for binary features.

<table>
<thead>
<tr>
<th></th>
<th>$E_i$</th>
<th>$E_j$</th>
<th>1 (presence)</th>
<th>0 (absence)</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (absence)</td>
<td>a</td>
<td>b</td>
<td>a + b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 (presence)</td>
<td>c</td>
<td>d</td>
<td>c + d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sum</td>
<td>a + c</td>
<td>b + d</td>
<td>n = a + b + c + d</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2.2.2. Similarity measures for non-binary features

Table 4 lists some well known similarity measures for non-binary features. In Table 4, $Ma$ represents the sum of features that are present in both entities $E_i$ and $E_j$, $Mb$ represents the sum of features that are present in $E_i$ but are absent in $E_j$, and $Mc$ represents the sum of features that are not present in $E_i$ and are present in $E_j$.

For software modularization, it has been shown that the Jaccard measure produces better results than other measures for binary features (Davey and Burt, 2000; Anquetil and Letherbridge, 1999). One reason for this is that it does not consider $d$ (absence of feature/negative match) (Maqbool and Babri, 2004; Saed et al., 2003). It has been observed that for software, the features are asymmetric, i.e., the presence of a feature, or ‘1’ has more weight than its absence, i.e., ‘0’. The absence of features does not indicate similarity between two entities, e.g., if two classes both do not use a variable, it does not mean that they are similar. For non-binary features, a counterpart of the Jaccard similarity measure called the ‘Unbiased Ellenberg’ produces better results for software clustering (Maqbool and Babri, 2007).

2.3. Application of a clustering algorithm

The next step is to apply a clustering algorithm. Clustering algorithms can be broadly categorized into hierarchical and partitional. Agglomerative Hierarchical Clustering (AHC) algorithms, which have been commonly employed for software clustering, are based on the bottom-up approach. In this approach, an algorithm considers each entity to be a singleton cluster and groups together the two most similar clusters at every step. At the end, the algorithm makes one large cluster which contains all the entities.

Partitional clustering produces flat groups with no hierarchy. In the software domain, partitional algorithms have also been used (Kanellopoulos et al., 2007; Lakhotia, 1997), but there are some advantages of using AHC algorithms. For example, AHC algorithms do not require prior information about the number of clusters. Moreover, the hierarchical structure of a software system is naturally represented through hierarchical algorithms. But the disadvantage is that we have to select a cutoff point, which represents the number of steps after which to stop the algorithm.

2.3.1. OUSM clustering algorithms

Widely used agglomerative hierarchical algorithms for software modularization which only update similarity matrix (OUSM) at every step of the clustering process (and do not change the feature matrix) are Complete Linkage, Single Linkage, Weighted Average Linkage, and Unweighted Average Linkage. When two entities are merged into a cluster, similarity between the newly formed cluster and other clusters/entities is calculated differently by these algorithms. Suppose we have three entities $E_1$, $E_2$, and $E_3$. Using these algorithms, similarity between $E_1$ and the newly formed cluster $E_23$ is calculated as (Anquetil and Letherbridge, 1999):

- **Complete**
  
  \[ \text{Similarity}(E_1, E_{23}) = \min(\text{Similarity}(E_1, E_2), \text{Similarity}(E_1, E_3)) \]

- **Single**

  \[ \text{Similarity}(E_1, E_{23}) = \max(\text{Similarity}(E_1, E_2), \text{Similarity}(E_1, E_3)) \]

- **Weighted Average**

  \[ \text{Similarity}(E_1, E_{23}) = \frac{1}{2} \times \text{Similarity}(E_1, E_2) + \frac{1}{2} \times \text{Similarity}(E_1, E_3) \]

- **Unweighted Average**

  \[ \text{Similarity}(E_1, E_{23}) = \frac{\text{Similarity}(E_1, E_2) \times \text{size}(E_2) + \text{Similarity}(E_1, E_3) \times \text{size}(E_3)}{\text{size}(E_2) + \text{size}(E_3)} \]

2.3.2. NFV clustering algorithms

Two hierarchical algorithms proposed for software modularization which make a new feature vector (NFV) of the newly formed cluster, are the Weighted Combined algorithm (WCom) (Maqbool and Babri, 2004) and the scalable Information Bottleneck (LinkBO) algorithm (Andritsos and Tzepros, 2005). When two entities are merged in a cluster, information about the number of entities accessing a feature is lost (Maqbool and Babri, 2004) when using linkage algorithms. The WCom algorithm and LinkBO overcome this limitation of linkage algorithms by making a new feature vector for the newly formed cluster. This feature vector contains information about the number of entities accessing a feature. Unlike linkage algorithms, these algorithms update the feature matrix after every step.

Suppose we have two entities $E_1$ and $E_2$ with normalized feature vectors $f_i$ and $f_j$, respectively. The new feature vector $f_{ij}$ is calculated for both the algorithms as:

\[ f_{ij} = \left( \frac{f_i + f_j}{n_i + n_j} \right) \]

where $n_i$ is the number of entities in ith cluster and $n_j$ is the number of entities in jth cluster.

It has been shown that the approach of updating the feature vector after every step in clustering produces better results in terms of authoritativeness (i.e., results are better because the automatically produced decomposition is more similar to the expert decomposition) (Maqbool and Babri, 2007) by retaining useful information.

---

Table 4
Similarity measures for non-binary features.

<table>
<thead>
<tr>
<th>S. no.</th>
<th>Name</th>
<th>Mathematical representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ellenberg</td>
<td>$0.5 \cdot Ma/(0.5 \cdot Ma + Mb + Mc)$</td>
</tr>
<tr>
<td>2</td>
<td>Unbiased Ellenberg</td>
<td>$0.5 \cdot Ma/(0.5 \cdot Ma + Mb + Mc)$</td>
</tr>
<tr>
<td>3</td>
<td>Gleason</td>
<td>$Ma/(Ma + Mb + Mc)$</td>
</tr>
<tr>
<td>4</td>
<td>Unbiased Gleason</td>
<td>$Ma/(Ma + b + c)$</td>
</tr>
</tbody>
</table>

Please cite this article in press as: Naseem, R., et al., Cooperative clustering for software modularization. J. Syst. Software (2013), http://dx.doi.org/10.1016/j.jss.2013.03.080
Algorithm 1. NFV algorithm

Input: \((n \times p)\) matrix

Output: Hierarchy of Clusters

1: Create a matrix by calculating similarity/distance between each pair of entities.
2: repeat
3: Merge the most similar entities into one cluster (using maximum value of similarity or minimum value in case of distance measures in similarity matrix).
4: Make a new feature vector \(f_k\) for the newly formed cluster in \((n \times p)\) matrix. Suppose we have two entities \(E_1\) and \(E_2\) with normalized feature vectors \(f_1\) and \(f_2\), respectively. The new feature vector \(f_k\) is calculated as:
\[
f_k = (f_1 + f_2)/(n_1 + n_2)
\]
where \(n_1\) is the number of entities in \(E_1\) and \(n_2\) is the number of entities in \(E_2\).
5: Update the similarity/distance matrix by recalculating similarity/distance between newly formed cluster and existing entities or clusters.
6: until the required number of clusters or a single large cluster is formed

The main steps in NFV algorithms are presented in Algorithm 1. In Algorithm 1, in case of the Weighted Combined algorithm, a measure for non-binary features must be used. The Unbiased Ellenberg measure has shown better results with this algorithm for software modularization (Maqbool and Babri, 2007) as compared to other measures. The Information Loss (IL) measure is used with the LIMBO algorithm to calculate the information loss between two entities/clusters. The entities are chosen for grouping together into a new cluster when their IL is minimum (for more detail see Andritsos and Tzerpos, 2005).

2.3.3. Assessment of results

In external assessment, the automatically prepared decompositions are compared with the decompositions prepared by human experts. For this purpose different measures may be used. A well known measure is MojoFM (Wen and Tzerpos, 2004a), a recent version of Mojo (Shern and Tzerpos, 2004). MojoFM is an external assessment measure which calculates the percentage of Move and Join operations to convert the decomposition produced by a clustering algorithm to an expert decomposition (Wen and Tzerpos, 2004a). To compare the result A of an algorithm with an expert decomposition B, we have:
\[
\text{MojoFM} = \left(1 - \frac{\text{mno}(A, B)}{\max(\text{mno}(A \cap B, B))}\right) \times 100
\]
where \(\text{mno}(A, B)\) is the minimum number of 'move' and 'join' operations needed to convert from A to B and \(\max(\text{mno}(A \cap B, B))\) is the maximum of the minimum number of possible 'move' and 'join' operations needed to convert from A to B. The value of MojoFM lies between 0% and 100%. A higher MojoFM value denotes greater correspondence between the two decompositions and hence better results, while lower MojoFM values indicate that the decompositions are different.

In internal assessment, some internal characteristic of clusters is used to evaluate the quality of results. Arbitrary decisions represent an internal quality measure (Maqbool and Babri, 2004). An arbitrary decision is taken by an algorithm when there is more than one maximum value for similarity between entities in the similarity matrix (or for distance and information loss measures, there is more than one minimum value). Thus the decision as to which two entities to cluster is arbitrary, since more than two entities are equally similar.

3. Feature vector cases

As described in Section 1, we define cooperative clustering to be the cooperation of more than one similarity measure during the clustering process. Thus to apply the cooperative technique in software clustering for modularization, our first step was to analyze similarity measures to identify their strengths and weaknesses. For this purpose, we 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) (Maqbool and Babri, 2007). In our preliminary work (Naseem et al., 2010, 2011), we identified some 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.

3.1. Feature vector case 1 (fvC1): value of \(a\) is different among entities, but Jaccard similarity is equal

An example feature matrix with four entities (E1–E4) and four features (f1–f4) for this case is presented in Table 5. From Table 5, 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 6. Table 6 can be seen from Table 6 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, i.e., it will choose E1, E2 or E3, E4 for clustering arbitrarily.\(^1\)

3.2. Feature vector case 2 (fvC2): value of \(a\) is high among entities, but they are not completely similar

An example feature matrix for fvC2 with four entities (E1–E4) and six features (f1–f6) is presented in Table 7. It can be seen that E1 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.

\(^1\) Here E1 and E2 share two features with E3 and E4. However, it is relevant to note that the same case also arises when E1 and E2 share no features with E3 and E4.

---

Table 5

<p>| Example A. |</p>
<table>
<thead>
<tr>
<th>Entities</th>
<th>f1</th>
<th>f2</th>
<th>f3</th>
<th>f4</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>E4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 6

<p>| Similarity measure using Jaccard for Example A. |</p>
<table>
<thead>
<tr>
<th>Entities</th>
<th>f1</th>
<th>f2</th>
<th>f3</th>
<th>f4</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>E2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>E3</td>
<td>0.5</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E4</td>
<td>0.5</td>
<td>0.5</td>
<td>1</td>
<td>–</td>
</tr>
</tbody>
</table>

Table 7

<p>| Example B. |</p>
<table>
<thead>
<tr>
<th>Entities</th>
<th>f1</th>
<th>f2</th>
<th>f3</th>
<th>f4</th>
<th>f5</th>
<th>f6</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>E4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

---

Please cite this article in press as: Naseem, R., et al., Cooperative clustering for software modularization. J. Syst. Software (2013), http://dx.doi.org/10.1016/j.jss.2013.03.080
The corresponding similarity matrix for Table 7 using the Jaccard measure is given in Table 8. It can be seen from Table 8 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 f6 which is accessed by E4 but not by E3.\(^2\) The feature vectors of E3 and E4 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 7 for \(fvc_2\), we can see that E3 and E4 have twice the number of common features as compared to E1 and E2, i.e., if \(a_1\) represents the number of common features for E1 and E2, and \(a_2\) represents the common features associated with E3 and E4, then E3 and E4 have \(a_2 = 2 \times a_1\). Value of \(b_2 + c_2 < a_1\) as can be seen in Table 7.

### 3.3. Discussion on the Jaccard measure in the light of \(fvc_1\) and \(fvc_2\)

Through \(fvc_1\) we showed that when entities share common features \((a>0)\) and \((b = c = 0)\), the Jaccard measure shows no difference in the similarity of these entities, i.e., it gives a similarity value = 1. This indicates that there is no difference between pairs of entities which share more 1's (E3 and E4 in Table 5) or share more 0's (E1 and E2 in Table 5). From this we conclude that for pairs of entities, if values of \(b\) and \(c\) = 0, their similarity will always be equal no matter what the value of \(a\), when using the Jaccard similarity measure.

Through \(fvc_2\), we showed that when \(b>0\) or \(c>0\), the Jaccard measure gives less importance to shared common features. So if two entities share 2 features only (E1 and E2 in Table 7) as compared to two entities sharing 4 features (E3 and E4 in Table 7), the Jaccard measure will consider the two entities sharing less features to be more similar if \(b>0\) or \(c>0\) for the other pair.

An analysis shows that both of the problems occur because the Jaccard similarity measure does not consider the proportion of total and shared features. The measure gives more importance to \(a\), while considering only \(b\) and \(c\) to determine proportion. It would be useful to define a measure that has the characteristics of the Jaccard measure (as this measure performs better than other measures for software modularization) but can solve the problems discussed above.

A study of the similarity measures that exist for binary features shows that a Jaccard-like similarity measure, Russell & Rao (Choi et al., 2010) exists, which considers the proportion of common and total number of features (see Table 2). The difference between the Russell & Rao measure \(a/(a+b+c+d)\) and the Jaccard measure \(a/(a+b+c+d)\) is the \(d\) in the denominator of Russell & Rao. The application of the Russell & Rao measure to \(fvc_1\) and \(fvc_2\) reveals that it overcomes deficiencies in the Jaccard similarity measure (see Appendix). However, the following case reveals a deficiency in the measure.

### 3.3.1. Feature vector case 3 (\(fvc_3\)): value of \(a\) is the same but values of \(b\) and \(c\) are not

Consider Table 9 having four entities (E1–E4) and five features (f1–f5). All the entities have the same value of \(a = 3\) but entities E1 and E2 have \(b = 1\) and \(c = 0\) while E3 and E4 have \(b = 1\) and \(c = 1\).

The corresponding similarity matrix using Russell & Rao is given in Table 10. It can be seen from Table 10 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 \(b = c = 0\).

### 3.4. Jaccard-NM – a new similarity measure for binary features

From the above discussion, we 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, we introduce a new Jaccard-like similarity measure 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 (Naseem et al., 2010):

\[
\text{Jaccard-NM} = \frac{a}{a + b + c + n}
\]

where \(n\) is the total number of features, i.e., \(n = a + b + c + d\), thus

\[
\text{Jaccard-NM} = \frac{a}{2(a + b + c + d)}
\]

Application of the Jaccard-NM measure to \(fvc_1\), \(fvc_3\) shows that it overcomes the deficiencies of the Jaccard measure for these cases, and its application to \(fvc_2\) shows that it overcomes the deficiency of the Russell & Rao measure (see Appendix).

### 3.5. Unbiased Ellenberg-NM – a new similarity measure for non-binary features

The cases \(fvc_1\), \(fvc_2\) and \(fvc_3\) can also occur in non-binary feature vectors. Therefore, to solve these problems, we proposed a new measure called Unbiased Ellenberg-NM. Our new measure is defined as follows (Naseem et al., 2011):

\[
\text{Unbiased Ellenberg-NM} = \frac{0.5 \times M_a}{0.5 \times M_a + b + c + n}
\]

### Table 8

Similarity matrix using Jaccard for Example B.

<table>
<thead>
<tr>
<th>Entities</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>E2</td>
<td>1</td>
<td>–</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>E3</td>
<td>0.4</td>
<td>0.4</td>
<td>–</td>
<td>0.6</td>
</tr>
<tr>
<td>E4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.6</td>
<td>–</td>
</tr>
</tbody>
</table>

### Table 9

Example C.

<table>
<thead>
<tr>
<th>Entities</th>
<th>f1</th>
<th>f2</th>
<th>f3</th>
<th>f4</th>
<th>f5</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>E4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

### Table 10

Similarity matrix using Russell & Rao for Example C.

<table>
<thead>
<tr>
<th>Entities</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>–</td>
<td>–</td>
<td>0.6</td>
<td>0.6</td>
</tr>
<tr>
<td>E2</td>
<td>0.6</td>
<td>–</td>
<td>0.6</td>
<td>–</td>
</tr>
<tr>
<td>E3</td>
<td>0.6</td>
<td>0.6</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>E4</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
<td>–</td>
</tr>
</tbody>
</table>

\(^2\) Similar to \(fvc_1\), \(fvc_2\) also arises when there are no shared features between E1, E2 and E3, E4.
Unbiased Ellenberg-NM = \[
\frac{0.5 \times Ma}{0.5 \times Ma + 2(b + c) + a + d}
\]
where \(Ma\) represents the sum of features that are shared between two entities, as explained in Section 2.2.2.

### 4. Cooperative software clustering

In the previous section, we discussed feature vector cases (fvcs) in \((n \times p)\) matrices, where the Jaccard measure does not perform well and our proposed similarity measure, i.e., Jaccard-NM produces better results. The main goal of analyzing the \((n \times p)\) matrix was 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, we perform 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". In this section, we describe our cooperative clustering approach and algorithms.

#### 4.1. Cases in features vectors of entities in the \((n \times p)\) matrix

In Section 3, we presented two cases fvc1 and fvc2. We can write fvc1 as:

\[
\text{fvc1} = (a > 0) \land (b + c) = 0
\]

Similarly we can write fvc2 as:

\[
fvc2 = \begin{cases} 
(a_1 > 0) \land (b_1 + c_1) = 0 \\
(a_2 = 2 \times a_1) \land (b_2 + c_2) \leq a_1
\end{cases}
\]

where \(a_1, b_1\) and \(c_1\) denote the \(a, b\) and \(c\) values in one pair of entities and \(a_2, b_2\) and \(c_2\) represent the values in the other pair of entities.

For non-binary features, they can be written as:

\[
fvc2 = \begin{cases} 
(Ma_1 > 0) \land (b_1 + c_1) = 0 \\
(Ma_2 = 2 \times Ma_1) \land (b_2 + c_2) \leq Ma_1
\end{cases}
\]

#### 4.2. Some 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 \(E_i\) and \(E_j\) then \((E_i, E_j) = (E_j, E_i)\) and \(i \neq j\). Entity \((E_i\) or\(E_j)\) may also represent a cluster containing more than one entity. For example, in Table 8 we have \((n \times n / 2)\), i.e., \((4 / 4 - 1 / 2) = 6\) similarity values. PESym are \((E2, E1)\) \((E3, E1)\) \((E4, E1)\) \((E3, E2)\) \((E4, E2)\) \((E4, E3)\).

**Definition 2.** The PESym-MaxSim is the Pair of Entities in the Symmetric Similarity Matrix Against Maximum Similarity. PESym-MaxSim is denoted by \(Ma\). For example, in Table 8 PESym-MaxSim is \((E2, E1)\) because the similarity value between \(E1\) and \(E2\) (1.0) is the maximum.

We also use the following notations. \(S_{\text{max}}\) and \(S_{\text{min}}\) are the maximum values in similarity matrix \(S_1\) and \(S_2\) respectively. \(c^*\) represents a new cluster \(J_{\text{sim}}\) is the Jaccard similarity measure and \(JNM_{\text{sim}}\) is the Jaccard-NM similarity measure. \(UE_{\text{sim}}\) represents the Unbiased Ellenberg similarity measure and \(UENM_{\text{sim}}\) represents the Unbiased Ellenberg-NM measure. \(NC\) is the Number of Clusters and \(LC\) is a Large Cluster.

#### 4.3. Cooperative OUSM software clustering

The main steps in the cooperative software clustering technique are given in Algorithm 2.

The Cooperative OUSM (COUSM) algorithm first creates a similarity matrix \(S_1\) for the PESym belonging to fvc1 and fvc2, using the Jaccard-NM measure. If no PESym belong to fvc1 or fvc2, it assigns a ‘0’ similarity value. Then the similarity matrix \(S_2\) is created for all the PESym in the given \((n \times p)\) matrix using the Jaccard measure. The algorithm then searches for the maximum similarity value in the similarity matrix \(S_1\) to make a cluster. If the similarity value in \(S_1\) is equal to ‘0’ then the algorithm searches for the maximum in \(S_2\). During this process, if the maximum is from similarity matrix \(S_1\), then the algorithm will update both similarity matrices \(S_1\) and \(S_2\) (Step 20). If the maximum value is from \(S_2\), then the algorithm will update only similarity matrix \(S_2\) (Step 22).

**Algorithm 2.** Cooperative OUSM algorithm

**Input:** \((n \times p)\) matrix.

**Output:** Hierarchy of Clusters.

```
1: for i = 1 to (n – 1) do
2: for j = 2 to n do
3: if \((E_i, E_j) \in \{fvc1 \ or \ fvc2\}\) then
4: \(S_1(i,j) = JNM_{\text{sim}}(E_i, E_j)\)
5: else
6: \(S_1(i,j) = 0\)
7: end if
8: \(S_1(i,j) = JNM_{\text{sim}}(E_i, E_j)\)
9: end for
10: end for
11: \(\text{bool} = \text{Notfinished}\)
12: repeat
13: if ((\(V_{\text{max}}^\text{sim}\) > 0) \&\& \(\text{bool} \neq \text{Finished}\)) then
14: \(c^* = Ma_{\text{max}}\)
15: else
16: \(c^* = Ma_{\text{sim}}\)
17: \(\text{bool} = \text{Finished}\)
18: end if
19: if (\(\text{bool} \neq \text{Finished}\)) then
20: Update \(S_1\) and \(S_2\), using basic linkage algorithm between PESym of \(c^*\) and \(E_i \neq c^*\)
21: else
22: Update \(S_2\), using basic linkage algorithm between PESym of \(c^*\) and \(E_i \neq c^*\)
23: end if
24: until Required NC or one LC is formed
```

#### 4.3.1. Illustration of COUSM algorithm

To illustrate our proposed algorithm for software clustering, we take a small example H with 8 entities and 13 features given in Table 11. Tables 12–22 show the similarity matrices \(S_1\) and \(S_2\) formed in various iterations. The clusters are shown as \((E_{ij})\) in a similarity matrix after each iteration. Step 4 creates the \(S_1\) similarity matrix using the Jaccard-NM similarity measure for PESym belonging to fvc1 or fvc2 else step 6 assigns ‘0’ similarity value as shown in Tables 12, 14, 16 and 18. Step 8 of the algorithm creates a similarity matrix \(S_2\) using the Jaccard similarity measure as given in Tables 13, 15, 17, 19–22.

---

Please cite this article in press as: Naseem, R., et al., Cooperative clustering for software modularization. J. Syst. Software (2013), http://dx.doi.org/10.1016/j.jss.2013.03.080
In the first iteration, steps 13 and 14 will select the maximum value from Table 12 and will check the condition for Boolean value (maximum value > 0) and will make a cluster of (E1, E2). Step 19 checks the Boolean condition. If not 'Finished', then both similarity matrices are updated by calculating similarity between the newly formed cluster c* and the existing entities or clusters (E_n - c*), using linkage algorithms (Tables 13–19). Bool remains not finished for 3 iterations. When the condition in step 19 becomes false then the algorithm will only update similarity matrix S2. For example see Tables 18 and 19, where in the 4th iteration, the maximum value in S1 (Table 18) is equal to '0', therefore the algorithm will jump to S2 (Table 19), to search for the maximum value in later iterations.

As per the proposed COUSM algorithm, the first cluster formed is (E1, E2), second is (E5, E6), third is (E3, E4), fourth is (E7, E8), fifth is ((E3, E4) (E5, E6)), sixth is ((E1, E2) (E7, E8)), and the last is ((E1, E2) (E7, E8))(((E3, E4) (E5, E6)). On the other hand, if we run the individual clustering algorithm using the Jaccard measure we get the clusters as (E1, E2) or (E3, E4) arbitrarily, (E5, E6), (E7, E8)), ((E3, E4) (E5, E6)), (E1, E2) (E7, E8), and ((E1, E2) (E7, E8))(E3, E4) (E5, E6)). Running the individual clustering algorithm using Jaccard-NM, we get the clusters as (E1, E2), (E5, E6), (E7, E8), ((E1, E2) (E7, E8)), (E3, E4), ((E3, E4) (E5, E6)), and ((E1, E2) (E7, E8))(E3, E4) (E5, E6)).
4.4. Cooperative software clustering using the WCCombined algorithm

The basic steps of our cooperative software clustering algorithm that integrate the Unbiased Ellenberg measure and the Unbiased Ellenberg-NM measure in the WCCombined algorithm are given in Algorithm 3. The steps for this algorithm are the same as for Algorithm 2. The only difference is in Step 17, where this algorithm uses the updation method of the \((n \times p)\) matrix by making a new feature vector for the new cluster and then calculates similarity for PESyM of the new cluster \((c^*)\) and existing clusters/entities.

**Algorithm 3.** Cooperative WCCombined (CWCombined) algorithm

**Input:** \((n \times p)\) matrix.

**Output:** Hierarchy of Clusters

1: \(\text{for } i = 1 \text{ to } (n - 1) \text{ do} \)
2: \(\text{for } j = 2 \text{ to } p \text{ do} \)
3: \(\text{if } (E_i, E_j) \in fV_1 \text{ or } fV_2 \text{ then} \)
4: \(S_i(j) = UENN_{nm}(E_i, E_j) \)
5: \(\text{else} \)
6: \(S_i(j) = 0 \)
7: \(\text{end if} \)
8: \(S_i(j) = UENN_{nm}(E_i, E_j) \)
9: \(\text{end for} \)
10: \(\text{end for} \)
11: \(\text{repeat} \)
12: \(\text{if } (V_{dhs} > 0) \text{ then} \)
13: \(c_* = M_{avg} \)
14: \(\text{else} \)
15: \(c_* = M_{std} \)
16: \(\text{end if} \)
17: \(\text{Update } (n \times p)\text{ matrix by taking} \)
18: \(f_k \left[ \left( \frac{1}{n_1 + n_2} \right) \right] = \left( \frac{1}{n_1 + n_2} \right) \)
19: \(\text{Update } S_0, \text{using } UENN_{nm}, \text{only between } \text{PESyM of } c^* \text{ and} \)
20: \(\text{until Required NC or one LC is formed} \)

5. Experimental setup

This section discusses the test systems and the clustering setup for our experiments.

5.1. The test systems

To conduct our clustering experiments, we selected four proprietary oriented software systems developed in Visual C++ (Abbasí, 2008) and one open source software, Mozilla\(^1\) developed using C++ and C languages. The proprietary software systems run under the Windows platform and are: (1) Statistical Analysis Visualization Tool is an application which provides functionality related to statistical data and result visualization. (2) Printer Language Converter is a part of another system, which provides conversion of intermediate data structures to a printer language. (3) Print Language Parser is a parser of a well-known printer language. It transforms plain text and stores the output in intermediate data structures. (4) Fact Extractor System is an extractor to parse Visual C++ systems and extract entities, features among entities and other related information. Mozilla is the open source Internet browser. We used Mozilla version 1.3.1 released in March 2003. Similar to the approach taken in Siddique and Maqbool (2012), we selected 6 subsystems out of 10 subsystems of Mozilla containing 258 Files. A brief description of the test systems is given in Table 23.

5.2. Fact extraction

We used the Fact Extractor System (Abbasí, 2008) to extract detailed design information, i.e., entities and relationships from the source code of the Visual C++ systems. Mozilla file relationships are taken from (Andreopoulos et al., 2007) and the process of fact extraction is explained in Andritsos and Tzerpos (2005).

5.3. Entities and features

Since all the proprietary systems are object oriented, we selected classes as entities for these systems. From the different relationships that exist between classes, we selected ten sibling (indirect) relationships (Muhammad, 2010) listed in Table 24. We selected indirect relationships because they can help capture semantic reasons for clustering entities (Maqbool and Babri, 2004; Anquetil and Lethbridge, 1999), and thus provide meaningful information during modularization of object oriented systems. Moreover, they have shown better results than direct relationships for software modularization (Muhammad et al., 2012, 2010).

For Mozilla, we selected files as entities because Mozilla comprises of both functions and classes within .c and .cpp files. Selecting only classes from .cpp files as entities will ignore the .c structure of the system, and selecting functions as entities will ignore the object oriented structure of the system, thus ignoring one language means ignoring a part of the system. The selected file based relationships for Mozilla are function calls (file F1 invokes a function declared in file F2), global access (a function declared in F1 refers to a global variable in F2) and data access (a function declared in F1 refers to a user defined type in F2).

We selected different types of entities (classes and files) because feature vector cases are related to the relationships between entities in a system and we wanted to use more than one entity type to see whether the identified feature vector cases arise in both cases.

5.4. Similarity measures

To find similarity between entities having binary features we selected the Jaccard, Jaccard-NM and Russell & Rao similarity measures. For non-binary features, we selected the Unbiased Ellenberg and Information Loss measures and compared their results with our proposed Unbiased Ellenberg-NM measure.

5.5. Algorithms

To cluster the most similar entities, we selected agglomerative clustering algorithms including Complete, Single, Weighted Average (Weighted) and Unweighted Average (Unweighted) described in Section 2.3.1. We also selected the WCCombined (Maqbool and Babri, 2004) and LIMBO (Andritsos and Tzerpos, 2005) described in Section 2.3.2.

5.6. Assessment

To evaluate the results, we used external assessment, i.e., we obtained an expert decomposition for each test system and compared our automatically produced clustering results with the expert decomposition. Since hierarchical clustering produces a decomposition at each step of the clustering process, a decision that must be taken is which decomposition to compare the expert decomposition with. An approach taken by researchers is to compare the expert decomposition with the automatically obtained decomposition at each step of the clustering process using a comparison measure and then take the average or maximum value (Maqbool and Babri, 2007). For our experiments, we followed a similar approach, i.e., we compared our clustering results with

---

the expert decomposition at every step of hierarchical clustering using the MojoFM measure (Wen and Tzerpos, 2004a). Results are reported by selecting the maximum MojoFM value obtained during the clustering process. To gain insight into the results, we also present the arbitrary decisions taken during the clustering process.

5.7. Expert decompositions

The expert decompositions for the four proprietary systems were developed by personnel having design and development experience in the software industry. For the Print Language Parser, we contacted an actual designer of the system to prepare the decomposition. For the other two systems, i.e., Statistical Analysis Visualization Tool and Fact Extractor, we selected the experts based on their skills and experience in the development of object-oriented systems. We provided the source code and class listing to all the experts and asked them to develop a decomposition of the given system. The experts were not provided with any details about clustering algorithms, and what relationships between entities were utilized during clustering. It is relevant to note that the expert decompositions for these systems have been utilized earlier for software modularization experiments in Muhammad (2010) and Siraj et al. (2012). For Mozilla, which is an open source system, the expert decomposition is taken from Siddique (2011). This expert decomposition has also been utilized earlier for software modularization experiments in Siddique and Maqbool (2012).

6. Experimental results

6.1. Results for the OUSM algorithms

In this section we describe the results of Cooperative OUSM (COUSM) and Individual OUSM (OUSM) algorithms for all data sets. Cooperative clustering for OUSM using Jaccard and Jaccard-NM measures is denoted as (J-JNM). Jaccard (J) and Jaccard-NM (JNM) indicate measures used in individual OUSM algorithms.

6.1.1. Feature vector cases and arbitrary decisions taken by the COUSM and OUSM algorithms

In Table 25, we present the statistics of $fv_{c1}$, $fv_{c2}$ and $fv_{c3}$ in all the test systems. These statistics reveal that the identified cases, especially $fv_{c1}$ and $fv_{c2}$, occur in large numbers during the clustering process. It can also be seen that occurrence of $fv_{c3}$ is much higher than that of $fv_{c1}$.

As described earlier, $fv_{c}$ results in arbitrary decisions by the algorithms. Table 26 presents the arbitrary decisions taken as a result of applying the COUSM and OUSM (using Jaccard, Jaccard-NM and Russell & Rao measures) algorithms throughout the clustering process for all the test systems.

From Table 26 we can see that for all the test systems, J and RR produce a larger number of arbitrary decisions as compared to J-JNM and JNM. The reason for this is the occurrence of $fv_{c1}$ at the start of the clustering process. RR produces more arbitrary decisions as compared to J-JNM and JNM due to the occurrence of $fv_{c3}$.

As an example consider the Print Language Parser system, which contains occurrences of $fv_{c1}$, $fv_{c2}$ and $fv_{c3}$. The number of arbitrary decisions taken by J, RR, JNM and J-JNM throughout the clustering process are presented in Fig. 1 for this system. It can be seen that J takes a larger number of arbitrary decisions in the first ten

Table 25

<table>
<thead>
<tr>
<th>Test systems</th>
<th>$fv_{c1}$</th>
<th>$fv_{c2}$</th>
<th>$fv_{c3}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistical Analysis Visualization Tool</td>
<td>123</td>
<td>18</td>
<td>1522</td>
</tr>
<tr>
<td>Printer Language Converter</td>
<td>362</td>
<td>0</td>
<td>564</td>
</tr>
<tr>
<td>Print Language Parser</td>
<td>60</td>
<td>3</td>
<td>1038</td>
</tr>
<tr>
<td>Fact Extractor System</td>
<td>6</td>
<td>0</td>
<td>363</td>
</tr>
<tr>
<td>Mozilla</td>
<td>28</td>
<td>5</td>
<td>3985</td>
</tr>
</tbody>
</table>

Please cite this article in press as: Naseem, R., et al., Cooperative clustering for software modularization. J. Syst. Software (2013), http://dx.doi.org/10.1016/j.jss.2013.03.080
iterations of the clustering process due to occurrence of $f_{vc1}$. Arbitrary decisions due to $f_{vc1}$ are reduced by both JNM and RR in the earlier iterations. However, RR also results in a larger number of arbitrary decisions as compared to other measures between iterations (13–31) due to the occurrence of $f_{vc2}$. It can be seen that at the end of the clustering process, all measures including J-JNM (OUSM) result in a large number of arbitrary decisions. This is because at the end of the clustering process, feature vectors have $a = 0.4$.

On the other hand, consider the Fact Extractor System, which contains only 6 $f_{vc}$ and no $f_{vc1}$. It can be seen from Fig. 1 that due to the small number of $f_{vc}$ and no $f_{vc1}$, the number of arbitrary decisions by J is reduced in the earlier iterations, whereas RR and JNM do not result in any arbitrary decision. The Fact Extractor System contains $f_{vc2}$ between iterations (13–15) and (20–24) due to which RR results in arbitrary decisions. Similar to the Print Language Parser, in the last half of the clustering process all the measures result in arbitrary decisions because of the feature vectors where $a = 0$.

For the Printer Language Converter System, the number of arbitrary decisions is considerably larger than the other systems, especially when using the J measure. The reason is that this system contains a larger number of $f_{vc2}$ as compared to the other systems. Our example in Fig. 1 shows that the effect of $f_{vc1}$ will be more pronounced in the first half of the clustering process, thus largely influencing the clustering results.

It can be seen from Table 26 that on average, JNM produces least number of arbitrary decisions, followed by J-JNM. Based upon the analysis of measures used in OUSM (Section 3), COUSM using J-JNM produces the expected number of arbitrary decisions. COUSM using J-JNM produces arbitrary decisions which are less than the J measure and are equal to or larger than the JNM measure for all systems. This is because for $f_{vc1}$ and $f_{vc2}$, we apply the JNM measure which reduces arbitrary decisions but for the rest of the PESyM, we apply the J measure which produces a larger number of arbitrary decisions as compared to JNM.

6.1.2. External evaluation of results for the COUSM and OUSM algorithms

Table 27 presents the results of the comparison between the automatically obtained decompositions by COUSM and OUSM, and the expert decomposition using MojoFM for all the test systems. We can see from Table 27 that the COUSM algorithm using J-JNM produces better results on average as compared to the OUSM algorithms using the J-JNM and RR measures for the Statistical Analysis Visualization Tool, Print Language Parser and Mozilla. These three datasets contain occurrences of $f_{vc1}, f_{vc2}$ and $f_{vc3}$. Thus our results reveal that when the identified feature vectors cases arise in software systems, cooperative clustering produces better results.

For the Printer Language Converter, it can be observed that the results are the same for J-JNM, J and RR, despite the fact that the arbitrary decisions taken by J are very large. To understand why this may be so, consider the example illustrated in Table 28 with 6 entities (E1–E6) and 4 features (f1–f4). In Table 28, for entities E1, E2 and E3, $a = 2$ and for entities E4, E5 and E6, $a = 4$. The corresponding similarity matrices using J and JNM are given in Tables 29 and 30, respectively. J produces six arbitrary decisions to cluster the entities, while JNM reduces these arbitrary decisions to three. Thus there is considerable reduction in the number of arbitrary decisions by JNM. However, given the similarity tables, there is a possibility that the clustering results are the same for both the measures in the case when entities E4, E5 and E6 are clustered before E1.
E2 and E3 arbitrarily by the Jaccard measure). A similar situation arises during clustering in the Printer Language Converter, resulting in a much larger number of arbitrary decisions by J, but almost the same clustering results (and hence MoJoFM values) for all the measures.

For the Fact Extractor System, results of RR are better than for the other measures. The reason for this may be the relatively small number of fvc3. Moreover, if the arbitrary decisions due to these fvc3 arise toward the end of the clustering process, the negative impact on the clustering quality is reduced.

It is interesting to note that although the JNM measure reduces arbitrary decisions as compared to the J-JNM measure (Table 26), results of the J-JNM measure are better in terms of authoritiveness. This supports earlier analysis that reducing arbitrary decisions is not the only factor to produce high quality results (Maqbool and Babri, 2007). Moreover, better results of J-JNM as compared to JNM indicate that after the three identified feature vector cases have been exhausted during the clustering process, certain cases are handled better by the J measure. One such case is illustrated in Table 31, which has four entities (E1–E4) with seven features (f1–f7)

From Table 31 it can be seen that for entities E1 and E2, a = 2 and b + c = 1, while for entities E3 and E4, a = 3 and b + c = 3. Thus the proportion of common features as compared to uncommon features is higher for E1 and E2. In this case, JNM finds E3 and E4 to be more similar than E1 and E2. On the other hand, J finds E1 and E2 to be more similar, which may be more appropriate. This example illustrates a case where using J instead of JNM may be more beneficial. It also highlights that no single measure may be appropriate for all situations that may arise during the clustering process, thus strengthening our case for cooperative clustering.

6.2. Results for NFV algorithms

This section presents results for Cooperative WCombined (CWCombined) algorithm using the Unbiased Ellenberg with Unbiased Ellenberg-NM (UE-UENM) measures, WCombined algorithm with Unbiased Ellenberg (UE) and Unbiased Ellenberg-NM (UENM) measures and Information Loss (IL) algorithm.
### Table 27
Experimental results using MoJoFM for Cooperative OUSM algorithms using Jaccard with Jaccard-NM (J-JNM) measures and individual OUSM algorithms using Jaccard (J), Jaccard-NM (JNM) and Russell & Rao (RR) measures for all data sets.

<table>
<thead>
<tr>
<th>Statistical Analysis</th>
<th>Print Language Parser</th>
<th>Visualization Tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>COUSM</td>
<td>OUSM</td>
<td>COUSM</td>
</tr>
<tr>
<td>J-JNM</td>
<td>JNM</td>
<td>RR</td>
</tr>
<tr>
<td>Complete</td>
<td>71</td>
<td>54</td>
</tr>
<tr>
<td>Single</td>
<td>63</td>
<td>32</td>
</tr>
<tr>
<td>Weighted</td>
<td>63</td>
<td>48</td>
</tr>
<tr>
<td>Unweighted</td>
<td>55</td>
<td>50</td>
</tr>
<tr>
<td>Average</td>
<td>63</td>
<td>46</td>
</tr>
</tbody>
</table>

### Table 28
Example D.

<table>
<thead>
<tr>
<th>Entities</th>
<th>f1</th>
<th>f2</th>
<th>f3</th>
<th>f4</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>E5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>E6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### Table 29
Similarity matrix using Jaccard for Example D.

<table>
<thead>
<tr>
<th>Entities</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>E5</th>
<th>E6</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E2</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E3</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E4</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E6</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### Table 30
Similarity matrix using Jaccard-NM for Example D.

<table>
<thead>
<tr>
<th>Entities</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>E5</th>
<th>E6</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E2</td>
<td>0.3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E3</td>
<td>0.3</td>
<td>0.3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E4</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E5</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E6</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.5</td>
<td>0.5</td>
<td>-</td>
</tr>
</tbody>
</table>

### Table 31
Example E.

<table>
<thead>
<tr>
<th>Entities</th>
<th>f1</th>
<th>f2</th>
<th>f3</th>
<th>f4</th>
<th>f5</th>
<th>f6</th>
<th>f7</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E4</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### Table 32
Statistics of fvc1, fvc2 and fvc3 in all data sets for CWCombined/WCombined algorithms.

<table>
<thead>
<tr>
<th>Measures</th>
<th>CWCombined</th>
<th>WCombined</th>
<th>LIMBO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistical Analysis Visualization Tool</td>
<td>3580</td>
<td>1167</td>
<td>41,614</td>
</tr>
<tr>
<td>Print Language Converter</td>
<td>5489</td>
<td>1727</td>
<td>15,903</td>
</tr>
<tr>
<td>Print Language Parser</td>
<td>440</td>
<td>1436</td>
<td>22,754</td>
</tr>
<tr>
<td>Fact Extractor System</td>
<td>10</td>
<td>8</td>
<td>4620</td>
</tr>
<tr>
<td>Mozilla</td>
<td>351</td>
<td>406</td>
<td>252,033</td>
</tr>
</tbody>
</table>

### Table 33
Average number of arbitrary decisions using Cooperative Weighted Combined (CWCombined) algorithm with UE-UENM measures, and Weighted Combined (WCombined) algorithm with UE and UENM measures for all data sets.

<table>
<thead>
<tr>
<th>Measures</th>
<th>CWCombined</th>
<th>WCombined</th>
<th>LIMBO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistical Analysis Visualization Tool</td>
<td>7</td>
<td>7</td>
<td>18</td>
</tr>
<tr>
<td>Print Language Converter</td>
<td>7</td>
<td>6.5</td>
<td>17.5</td>
</tr>
<tr>
<td>Print Language Parser</td>
<td>2.7</td>
<td>2.3</td>
<td>3.9</td>
</tr>
<tr>
<td>Fact Extractor System</td>
<td>2.9</td>
<td>2.9</td>
<td>3</td>
</tr>
<tr>
<td>Mozilla</td>
<td>3</td>
<td>2.2</td>
<td>4.3</td>
</tr>
<tr>
<td>Average</td>
<td>4.52</td>
<td>4.18</td>
<td>9.34</td>
</tr>
</tbody>
</table>

6.2.1. Feature vector cases and arbitrary decisions for CWCombined, WCombined and LIMBO algorithms

Table 32 presents statistics of fvc1, fvc2 and fvc3 in the test systems. It is relevant to note that these statistics are different from those presented earlier in Table 25 because NNF algorithms update the feature vectors at every step of the clustering process.

The average number of arbitrary decisions for CWCombined algorithm using the UE-UENM measures and WCombined algorithm using the UE, UENM measures and IL algorithm are presented in Table 33. It was expected that the number of arbitrary decisions by UE-UENM for WCombined would be less than for other similarity measures. The experimental results confirm our expectations. On average we can see that UENM and UE-UEMN produce less number of arbitrary decisions as compared to UE and IL.

6.2.2. External evaluation of results for CWCombined, WCombined and LIMBO algorithms

Table 34 presents results of the comparison between the automatically obtained decomposition using CWCombined and

### Table 34
Experimental results using MoJoFM for all data sets using Cooperative Weighted Combined algorithm (CWCombined) with UE-UENM measures and Weighted Combined (WCombined) algorithm using UE and UENM measures.

<table>
<thead>
<tr>
<th>Measures</th>
<th>CWCombined</th>
<th>WCombined</th>
<th>LIMBO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistical Analysis Visualization Tool</td>
<td>74</td>
<td>68</td>
<td>74</td>
</tr>
<tr>
<td>Print Language Converter</td>
<td>68</td>
<td>68</td>
<td>71</td>
</tr>
<tr>
<td>Print Language Parser</td>
<td>76</td>
<td>70</td>
<td>73</td>
</tr>
<tr>
<td>Fact Extractor System</td>
<td>50</td>
<td>50</td>
<td>42</td>
</tr>
<tr>
<td>Mozilla</td>
<td>73</td>
<td>73</td>
<td>73</td>
</tr>
<tr>
<td>Average</td>
<td>68.2</td>
<td>65.8</td>
<td>66.6</td>
</tr>
</tbody>
</table>

---

Please cite this article in press as: Naseem, R., et al., Cooperative clustering for software modularization. J. Syst. Software (2013), http://dx.doi.org/10.1016/j.jss.2013.03.080
WCombined algorithms, and the expert decomposition using MoJoFM for all test systems.

Table 34 indicates that for all data sets, UENM gives better results as compared to UE and IL for all test systems except the Fact Extractor System. It can be seen from Table 34 that for the Fact Extractor, UENM produces relatively poor results. The reason for this may be that this system contains a small number of $f_{c1}$ and $f_{c2}$ as can be seen from Table 32. We can see from Table 34 that on average, CWCombined produces better results thus indicating that cooperative clustering improves results for NFV algorithms in a manner similar to OUSM algorithms.

6.3. Summary of results

From the results presented, we can conclude that the feature vector cases we have discussed arise in large numbers in software systems, especially when the NFV algorithms are used. These $f_{c1}$ results in arbitrary decisions by the clustering algorithms. Experiments reveal correspondence between the number of $f_{c1}$ and $f_{c2}$, and the number of arbitrary decisions taken by the Jaccard measure when using OUSM algorithms. The proposed JNM measure reduces arbitrary decisions as compared to the other measures.

External evaluation reveals that the cooperative approach using the J-JNM measure produces clusters closest to the expert decomposition for OUSM algorithms, although the number of arbitrary decisions taken by J-JNM is more than those taken by JNM. This supports earlier results that although reducing arbitrary decisions is beneficial, it is not the only factor in producing high quality results. It can also be seen that the best modularization results are obtained for the test systems when using the cooperative approach, for both the OUSM and NFV algorithms.

7. Threats to validity

In this section, we discuss factors that may affect the validity of our approach.

Internal validity: Our analysis in the paper is based on feature vector cases that may be present in the feature vector matrix of a software system, or may arise during the clustering process. These cases are general, and do not depend on the application domain characteristics of a software system. Thus the threat that domain characteristics of a software system are causing better cooperative clustering results is reduced. However, results depend on the frequency of occurrence of the various feature vector cases, and thus may vary for different systems. For our experiments, we selected proprietary as well as open source software systems. Moreover, we selected different types of entities (classes and files). This was done to allow analysis of the effect of these variations on the modularization results.

External validity: We conducted experiments on four proprietary systems written in C++ and an open source software system written in C++ and C. These systems vary in their applications and size. A threat to validity may arise due to the fact that we have considered systems written in C++ and C, which have certain relationships that may not be present in systems written using other languages (e.g., the relation ‘same file’ may not exist in systems written in Java). Therefore, to generalize our results, we need to consider systems in other programming languages, e.g., Java.

In a study by Glorie et al. (2009) in which formal concept analysis and clustering are applied to split a large monolithic software system into sub-systems, the conclusion is that both techniques scale badly for large industrial software. In their approach, the genetic algorithm provided by the Bunch tool (Mancoridis et al., 1999) is used for clustering. Glorie et al. reported that it sometimes took 10 h or more to produce results. Hierarchical clustering algorithms do not take a search based approach. They do not require multiple runs, and finish clustering in $n-1$ steps where $n$ represents the number of entities in a software system. The cooperative clustering approach that we propose also finishes in $n-1$ steps, although more computation is required per step till the identified cases exist in the feature vector matrix of a software system. Thus it is expected that the cooperative algorithm will scale as well as individual hierarchical clustering algorithms.

We acknowledge that our test systems do not have a very large size. On average, our test systems contain 71 classes. Although this is reasonable because in real projects during maintenance, developers perform their task on a small part of the whole system (Abbes et al., 2011), however experiments may be performed on test systems with larger sizes to empirically verify the scalability of our approach.

Construct validity: We used well known similarity measures, clustering algorithms and evaluation measures that have been employed for modularizing software. To determine similarity between entities, we used a number of relationships that may exist between them. We selected hierarchical clustering algorithms to allow cooperation between similarity measures during algorithm execution. External evaluation was carried out using the MoJoFM measure (Wen and Tzerpos, 2004a). We acknowledge that a threat to validity may exist due to the involvement of human experts during preparation of expert decompositions with which the automatically developed decomposition is compared. We have tried to reduce this threat by selecting experienced software personnel as experts, who were also involved in the development of the proprietary test systems. Moreover, the expert decompositions we selected have been used earlier in software modularization experiments in Siraj et al. (2012) and Siddique and Maqbool (2012).

8. Related work

Combining multiple clustering algorithms may be used to overcome the deficiencies in individual algorithms and has been shown to be effective in improving the cluster quality (Kashef and Kamel, 2007a). This combination of techniques has been defined in different ways by different researchers, e.g., ensemble (Strehl and Ghosh, 2003), collaborative (Forestier et al., 2010; Mitra et al., 2005), cooperative (Kashef and Kamel, 2010, 2009, 2007b), and cascaded (Qian and Suen, 2000).

Cooperative clustering has not been applied for software modularization. However, some hybrid approaches have been used to cluster software systems. For example, in Mamaghan and Meybodi (2009), a new hybrid evolutionary algorithm obtained by combining learning automata and genetic algorithms is proposed. This technique is more stable than the individual algorithms and speeds up the searching process. Another hybrid clustering approach based on combining graph clustering and partitioning is proposed in Sora et al. (2010). Zhang et al. defined the WDCG (Weighted Directed Class Graph) for object oriented software clustering (Zhang et al., 2010). In the WDCG, non-binary features are represented by edge weights and classes are vertices (entities). Zhang et al. proposed a new hybrid algorithm to recover the module structure from the WDCG of a software system.

A comprehensive overview of traditional individual clustering algorithms is presented in Wiggerts (1997). Wiggerts concluded that clustering techniques are appropriate for remodularization because they gather similar entities into groups, which is also the purpose of modularization. He also discussed the issue of whether the absence of a feature (0-0 match) effects the value of a similarity measure positively. In Anquetil and Lethbridge (1999) and Anquetil and Lethbridge (2003), four basic agglomerative hierarchical algorithms including Complete, Single, Weighted and Unweighted are...
evaluated. From experimental results it was concluded that Complete produces clusters which are more cohesive as compared to other algorithms.

In Patel et al. (2009) a novel approach is proposed, in which dynamic information is incorporated to build a skeletal decomposition of a software system by taking a specific scenario suggested by the user. The skeletal decomposition is then enriched with static information obtained from the source code. In Maqbool and Babri (2007), Maqbool and Babri present a comprehensive review of Hierarchical Clustering Algorithms for software clustering. They also analyze the behavior of similarity and distance measures for binary and non-binary features. In Wang et al. (2010), Wang et al. introduce an improved hierarchical clustering algorithm LBHFCL (LIMBO Based Fuzzy Hierarchical Clustering). They concluded that LBHFCL produces results having a larger number of clusters and less number of arbitrary decisions.

To obtain a layered architecture of a software system, in Andreopoulos et al. (2005) and Andreopoulos et al. (2007), an algorithm called MULICsoft is proposed. MULICsoft is a clustering algorithm that uses both static and dynamic information. Recently, Kleinberg algorithm was applied in Scanniello et al. (2010) to group entities into layers to recover the layered architecture of a software system. Constantinou et al. (2011) analyzed the correlation of design metrics with the layered architecture and investigated design metrics for layered architecture recovery using graph clustering techniques. Experimental results on two open source software systems (OpenProj and Rhino) were promising. Han et al. (2009) proposed a novel approach for clustering open source software systems using design patterns. To detect design patterns, a tool (Tastel) was developed. Experimental results indicated that Tastel is a more promising tool than Bunch.

Lungu et al. (2012) present a software architecture recovery tool called SoftwareNet. SoftwareNet has the capability to cluster a system and recover its architecture through interactive exploration and visualization. Muhammad et al. (2010, 2012) evaluated a large number of relationships among classes for clustering object oriented software systems. They implemented well known hierarchical clustering algorithms, i.e., Complete, Weighted, Unweighted, and also Bunch and ACDC (Tzerpos and Holt, 2000) for clustering nine different software systems. Experimental results showed that indirect relationships produce better results as compared to direct relationships.

Besides the agglomerative hierarchical algorithms used for software clustering, other techniques have also been applied for modularization of software systems. In Doval et al. (1999), a software system is presented as a Module Dependency Graph (MDG). An MDG is a directed graph in which entities are represented as nodes, and relationships among entities are represented as edges. Thus software modularization can be modeled as a graph partitioning problem, which minimizes coupling (inter edges) and maximizes cohesion (intra edges) (Harman et al., 2002). This approach results in a large space of solutions that needs to be searched for an optimal result (Harman et al., 2012), making it an interesting problem for search based software engineering (SBSE).

In this regard, Mancoridis et al. (1998) developed a tool called Bunch and used a hill-climbing algorithm with a single objective, i.e., modularization quality (MQ), to modularize software systems. The Bunch tool was enhanced over time to include new features (Mancoridis et al., 1999; Mitchell and Mancoridis, 2002). A detailed discussion on search based software modularization and the Bunch clustering tool can be found in Mitchell and Mancoridis (2006, 2008). Mahdavi et al. (2003) and Harman et al. (2005) also applied the hill climbing approach used in Bunch and re-implemented it in a tool called BruBunch. Other search based techniques, e.g., simulated annealing have been used for software modularization. However, experimental results indicate that hill-climbing algorithms produce better results in reasonable time (Harman et al., 2002; Mahdavi et al., 2003).

In SBSE, earlier work, e.g., Mancoridis et al. (1998, 1999) and Mahdavi et al. (2003), used a single-objective formulation for software clustering, i.e., MQ, which contains the twin objectives of high cohesion and low coupling. In Praditwong et al. (2011), a novel approach is suggested to solve the modularization problem. Instead of the single objective approach employed by search based approaches (Doval et al., 1999; Mancoridis et al., 1998), the paper suggests that multiple objectives be used. These multiple objectives are combined in (i) MCA which uses a set of five objectives (maximizing intra-cluster edges, number of clusters and MQ (Mitchell and Mancoridis, 2006), minimizing inter-cluster edges and number of isolated clusters), and (ii) ECA which uses the same set of objectives except minimizing the number of isolated clusters. This objective is replaced with the objective of minimizing the difference between maximum and minimum number of modules in a cluster. Comparison with the hill climbing approach of Bunch (Mancoridis et al., 1998), shows that the multi-objective approach outperforms the single objective approach for weighted graphs even when the evaluation criteria is based on the single objective used by the single objective approach.

Deepika and Brindha (2012), Kishore and Srinivasulu (2012) and Kishore et al. (2012) also reported that the multi-objective approach outperforms the single-objective approach. Barros et al. (2012) analyzed the effect of ECA and MQ in multi-objective software clustering. From experimental results, the author concluded that by suppressing MQ, multi-objective genetic algorithms can produce high quality solutions using less processing time. Although Praditwong et al. (2011) also solve the modularization problem, there are major differences between their approach and our approach. Firstly, we do not treat the problem as a search based problem, and thus do not employ evolutionary algorithms. We use hierarchical agglomerative clustering algorithms (Anquetil and Lethbridge, 1999) which take a deterministic approach. Multiple objectives are used in Praditwong et al. (2011), and we also use more than one similarity measure. Although the aim of both approaches is to produce a good modularization, this is more clearly defined in terms of cohesion and coupling in the search based approach. In our paper, evaluation of a good modularization is on the basis of external assessment, i.e., comparison of the automatically produced modularization with one developed by a human expert. Moreover, in our approach we define an algorithm which combines more than one similarity measure during the clustering process. Different similarity measures are used by our cooperative algorithm depending on the situation (or cases) that arise during clustering. The search based approach on the other hand combines the multiple objectives together to present a solution by considering all objectives at one time.

It may be interesting to apply the cooperative approach based on hierarchical clustering in this paper and the multi-objective search based approach of Praditwong et al. (2011) to the same problem to compare them empirically.

9. Conclusions

In the software domain, an important application of cluster analysis is to modularize a software system by grouping together software entities that are similar or related to each other. During maintenance, most of the effort is usually devoted to understanding a software system. This task is facilitated if a system is well modularized, making it easier to change it and also to evaluate the side effects of a change. A number of clustering algorithms have been employed for software modularization. To combine the strengths of various algorithms, researchers have proposed the idea of allowing
cooperation between them. The Cooperative Clustering Technique (CCT) 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.

In this paper, we investigate whether CCT can be applied for software modularization to improve results. Three research questions were formulated in this context.

(1) Can more than one similarity measure be made to cooperate during the clustering process?

We proposed a CCT in which two similarity measures cooperate during the hierarchical clustering process. For this purpose, we presented an analysis of well-known measures to identify their strengths and weaknesses. We identified deficiencies in the Jaccard similarity measure (which has shown better results than other measures for software modularization). We also described a new Jaccard-like measure called Jaccard-NM similarity measure which overcomes the identified deficiencies in the existing measures. For non-binary features, we discussed our Unbiased Ellenberg-NM measure, the non-binary counterpart of Jaccard-NM measure. From our analysis we conclude that individual measures show weaknesses in certain cases, and no single measure gives better results than others in all cases. Thus, more than one similarity measure can be made to cooperate during the clustering process to utilize their strengths and overcome their weaknesses.

(2) Is there a relation between arbitrary decisions taken during the clustering process and the modularization results?

During analysis of similarity measures, we identified two cases (fvc1 and fvc2) which revealed weaknesses of the Jaccard measure. Our experiments show correspondence between the number of fvc1 and fvc2 in test systems and the arbitrary decisions taken by the Jaccard measure. However, a reduction in arbitrary decisions does not guarantee better modularization results. Although the proposed Jaccard-NM measure reduces arbitrary decisions as compared to other measures, it does not give better results than the cooperative approach.

(3) Does the cooperative technique improve results for different types of hierarchical clustering algorithms for binary as well as non-binary features?

For the basic traditional clustering algorithms, we proposed a new Cooperative Only Update Similarity Matrix (COUSM) technique for software modularization. The COUSM technique integrates the binary Jaccard and Jaccard-NM similarity measures in a clustering process using well-known linkage algorithms. For clustering algorithms proposed specifically for software modularization, we proposed a new CCT called the Cooperative Weighted Combined (CWCombined) technique. This technique integrates the non-binary Unbiased Ellenberg and Unbiased Ellenberg-NM measures in the Weighted Combined (WCombined) algorithm. The modularization results of our cooperative approach for both COUSM and CWCombined show greater correspondence with the decompositions produced by human experts on average for various clustering algorithms and test systems. For example, for the Statistical Analysis Visualization Tool system, the average MultiOM value for COUSM is 63%, which is much higher than the 46% value produced by an individual clustering algorithm when using the Jaccard measure.

These results highlight that a single similarity measure may not appropriately handle the various situations that arise during the software clustering process. Cooperation between measures is thus a promising approach to make use of the strengths of the various measures during clustering. This is revealed by our experiments where the cooperative approach

<table>
<thead>
<tr>
<th>Entities</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E2</td>
<td>0.5</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E3</td>
<td>0.5</td>
<td>0.5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E4</td>
<td>0.5</td>
<td>0.5</td>
<td>1</td>
<td>-</td>
</tr>
</tbody>
</table>

Table A35: Similarity matrix using Russell & Rao for Example A.

<table>
<thead>
<tr>
<th>Entities</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E2</td>
<td>0.33</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E3</td>
<td>0.33</td>
<td>0.33</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E4</td>
<td>0.33</td>
<td>0.33</td>
<td>0.6</td>
<td>-</td>
</tr>
</tbody>
</table>

Table A36: Similarity matrix using Russell & Rao for Example B.

<table>
<thead>
<tr>
<th>Entities</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E2</td>
<td>0.3</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E3</td>
<td>0.25</td>
<td>0.25</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E4</td>
<td>0.25</td>
<td>0.25</td>
<td>0.5</td>
<td>-</td>
</tr>
</tbody>
</table>

Table A37: Similarity matrix using Jaccard-NM for Example A.

The application of the Russell & Rao measure to fvc1 and fvc2 reveals that it overcomes the deficiency of the Jaccard measure for these cases.

Appendix A.

A.1. The Russell & Rao measure and feature vector cases

We apply Russell & Rao to the feature vector cases, fvc1 and fvc2, to check whether it overcomes deficiencies of the Jaccard measure.

- fvc1 and the Russell & Rao measure

Table A35 presents the similarity matrix for Russell & Rao for system A (Table 5). We can see from Table A35 that Russell & Rao prioritizes the similarity between entities E3 and E4, E1 and E2. Thus, the decision to cluster entities is no longer arbitrary; E3 and E4 are most similar and will be grouped first.

- fvc2 and Russell & Rao measure

Table A36 is the corresponding similarity matrix for Example B (Table 7), using the Russell & Rao measure. It can be seen from Table A36 that the Russell & Rao measure finds E3 and E4 to be more similar, as was suggested, because E3 and E4 share more features (higher value of a) as compared to E1 and E2.

The application of the Russell & Rao measure to fvc1 and fvc2 reveals that it overcomes the deficiency of the Jaccard measure for these cases.

A.2. The feature vector cases and the Jaccard-NM measure

We apply Jaccard-NM to the cases discussed above, i.e., two cases for Jaccard (fvc1 and fvc2) and one case for Russell & Rao (fvc3).

- fvc1 and the Jaccard-NM measure

The similarity matrix using Jaccard-NM for fvc1 in Table 5 is given in Table A37. From Table A37 we can see that Jaccard-NM prioritizes the similarity values of (E4, E3) and (E2, E1), and creates no arbitrary decision for these pair of entities. Jaccard-NM finds (E4, E3) to be most similar, as should have been the case.

- fvc2 and the Jaccard-NM measure

Table A38 is the similarity matrix using the Jaccard-NM measure for System B given in Table 7. The Jaccard measure finds

Please cite this article in press as: Naseem, R., et al., Cooperative clustering for software modularization. J. Syst. Software (2013), http://dx.doi.org/10.1016/j.jss.2013.03.080
(E4, E3) to be more similar even though this pair contains non-zero values for b and/or c. As was suggested this measure give less importance to the pair of entities (E2, E1), because this pair shares less number of features as compared to (E4, E3).

- $f_{vc_1}$ and the Jaccard-NM measure

The similarity matrix using Jaccard-NM for $f_{vc_1}$ in Table 9 is given in Table A39. From Table A39 we can see that Jaccard measure produces no arbitrary decisions for pairs of entities (E2, E1) and (E4, E3) (unlike the Russell & Rao measure), and thus finds (E2, E1) to be most similar.

The application of the Jaccard-NM measure to $f_{vc_1}$ and $f_{vc_2}$ reveals that it overcomes the deficiency of the Jaccard measure for these cases. Moreover, its application to $f_{vc_3}$ reveals that it also overcomes deficiency of the Russell & Rao measure for this case.

A.3. Presence of d’ zero-dimension and similarity/distance measures

It is interesting to note that the Jaccard-NM and Unbiased Ellenberg-NM measures consider d, i.e., the absence of features. Previous research indicates that measures considering d do not give good results, because absence of features does not indicate similarity (Anquetil and Lethbridge, 1999; Saeed et al., 2003). It can be seen from Table A2 that similarity measures containing d consider it in the numerator (a sign of similarity) as well as in the denominator. However, in our case, we do not consider d to be a sign of similarity between two entities, rather it is included in the denominator only, for determining the proportion of common features as compared to total features.

To show that the presence of d in a measure do not necessarily deteriorate results, consider Table A40 which shows 4 entities (E1–E4) and 7 features (f1–f7). E1 and E2 share two features, so that value of a is 2. Both of them access one feature each, that the other entity does not, so $b = 1$ and $c = 1$. E3 and E4 share three features, so $a = 3$. Similar to E1 and E2, both of them access one feature each that the other entity does not, so $b = 1$ and $c = 1$.

The similarity matrix according to the Jaccard measure is given in Table A41. The similarity matrices according to the Jaccard-NM, Russell & Rao and Simple Matching measures (all of which contain d) are given in Tables A42–A44. It can be seen from Tables A41–A43 that the Jaccard, Jaccard-NM and Russell & Rao measures find E3 and E4 to be most similar. From Table A40, it is clear that E3 and E4 should indeed be considered most similar. However, due to the presence of d in the numerator of the Simple Matching coefficient, it finds E1 & E2 and E3 & E4 to be equally similar, resulting in an arbitrary decision where either of these entities may be grouped.

From this example, it is clear that the significant factor is whether d is present in the numerator or denominator of a measure. Its presence in the numerator deteriorates results (as for the Simple Matching coefficient). However, if it is present in the denominator only, it does not indicate similarity but it is a useful indicator of the proportion of common and total features.

References


Please cite this article in press as: Naseem, R., et al., Cooperative clustering for software modularization. J. Syst. Software (2013), http://dx.doi.org/10.1016/j.jss.2013.03.080
an assistant professor. Her research interests include software reverse engineering, architecture recovery, and machine learning.

Siraj Muhammad received the M.Phil degree in computer science from the Quaid-i-Azam University, Pakistan, in 2010. He was with the software industry for some years. He joined the Department of Computer Science, Shaheed Benazir Bhutto University, Sheringal, Dir(U), as a teaching faculty in 2011, where he is currently an assistant professor and head of the department. His research interests include software reverse engineering, architecture recovery, and clustering techniques.