Available online at www.sciencedirect.com
ScienceDirect
Information and Software Technology 49 (2007) 244–254
INFORMATION AND SOFTWARE TECHNOLOGY
www.elsevier.com/locate/infsof

Clustering large software systems at multiple layers

Bill Andreopoulos [a,*], Aijun An [a], Vassilios Tzerpos [a], Xiaogang Wang [b]

[a] Department of Computer Science and Engineering, York University, Toronto, Ont., Canada M3J1P3
[b] Department of Mathematics and Statistics, York University, Toronto, Ont., Canada M3J1P3

Received 5 July 2006; accepted 25 October 2006
Available online 8 December 2006

Abstract

Software clustering algorithms presented in the literature rarely incorporate in the clustering process dynamic information, such as the number of function invocations during runtime. Moreover, the structure of a software system is often multi-layered, while existing clustering algorithms often create flat system decompositions.

This paper presents a software clustering algorithm called MULICsoft that incorporates in the clustering process both static and dynamic information. MULICsoft produces layered clusters with the core elements of each cluster assigned to the top layer. We present experimental results of applying MULICsoft to a large open-source system. Comparison with existing software clustering algorithms indicates that MULICsoft is able to produce decompositions that are close to those created by system experts.

Keywords: Software clustering; Multiple layer; Categorical; MULIC; Graph

1. Introduction

Reverse engineering is the process of analyzing a system's internal elements and its external behavior and creating a structural view of the system. Automatic construction of a structural view of a large legacy system can significantly facilitate the developers' understanding of how the system works. In legacy systems, the original source code is often the only available source of information about the system and it is very time consuming to study.

Software clustering techniques aim to decompose a software system into meaningful subsystems, to help new developers understand the system. Clustering is applied to large software systems in order to partition the source files of the system into clusters, such that files containing source code with similar functionality are placed in the same cluster, while files in different clusters contain source code that performs dissimilar functions. Software clustering can be done *automatically* or *manually*. Automatic clustering of a large software system using a clustering tool is especially useful in the absence of experts or accurate design documentation. It is desirable to have a software clustering tool that can consider both static and dynamic system information. Automatic clustering techniques generally employ certain criteria (i.e., low coupling and high cohesion) in order to decompose a software system into subsystems [13,12,17]. Manual decomposition of the system is done by software engineers. However, it is time consuming and it requires full knowledge of the system.

We propose the MULICsoft software clustering algorithm that is based on the MULIC categorical clustering algorithm that is described in [2]. MULICsoft differs from MULIC in that it incorporates both static and dynamic information (i.e., the number of function calls during runtime) in the software clustering process. MULICsoft handles dynamic information by associating weights with file dependencies and incorporating the weights in the clustering process through special similarity metrics. We showed that MULIC clustering results are of higher quality than those of other categorical clustering algorithms, such as *k*-Modes, ROCK, AutoClass, CLOPE and others [2]. Characteristics

* Corresponding author.
E-mail addresses: billa@cse.yorku.ca (B. Andreopoulos), aan@cs.yorku.ca (A. An), bil@cse.yorku.ca (V. Tzerpos), stevenw@mathstat.yorku.ca (X. Wang).

0950-5849/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2006.10.010

of MULIC and MULICsoft include: *a*. The algorithm does not sacrifice the quality of the resulting clusters for the number of clusters desired. Instead, it produces as many clusters as there naturally exist in the data set. *b*. Each cluster consists of layers formed gradually through iterations, by relaxing the similarity criterion for inserting objects (files) in layers of a cluster at different iterations.

Section 2 gives an overview of previous software clustering tools. Section 3 describes the formulation of the input data for our clustering approach. Section 4 describes the MULICsoft clustering algorithm. Section 5 describes experimental results on the Mozilla system. Section 6 discusses inputting additional data to MULICsoft. Section 7 discusses evaluation of the results using an alternative measure. Section 8 discusses the runtime performance. Finally, Section 9 concludes and discusses future work.

2. Related work

Several clustering algorithms for software have been presented in the literature [3,5,6,9,11,12,14,15,17,20]. Some of the existing software clustering tools, such as LIMBO [3], consider dynamic information (i.e., the number of function calls during runtime) in the clustering process, while others, such as Bunch [12] and ACDC [17], produce clusters with a nested structure. MULICsoft both considers dynamic information and produces clusters with a layered structure.

In this section, we describe three algorithms: Bunch [12], ACDC [17] and LIMBO [3,4]. In Section 5, we will compare our proposed algorithm to these established software clustering algorithms.

Bunch is a clustering tool intended to aid the software developer and maintainer in understanding, verifying and maintaining a source code base [12]. The input to Bunch is a module dependency graph (MDG). Fig. 1 shows an MDG graph. Bunch views the clustering problem as trying to find a good partition of an MDG graph. Bunch defines a "good partition" as a partition where highly interdependent modules are grouped in the same cluster (representing subsystems) and independent modules are assigned to separate clusters. Fig. 1b shows a "good" partitioning of Fig. 1a. Finding a good graph partition involves systematically navigating through a very large search space of all possible partitions for that graph. Bunch treats graph partitioning (clustering) as an optimization problem. The goal of the optimization is to maximize the value of an objective function, called modularization quality (MQ) [12].

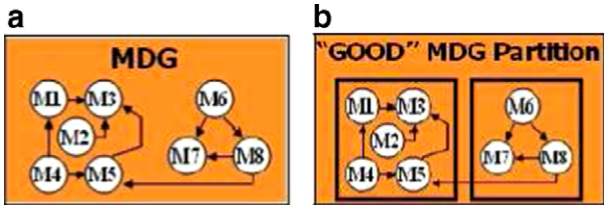

Fig. 1. (a and b) An MDG graph and its partition of maximum MQ [12].

ACDC works in a different way from other algorithms. Most software clustering algorithms identify clusters by utilizing criteria such as the maximization of cohesion, the minimization of coupling, or some combination of the two. ACDC performs the task of clustering in two stages. In the first stage, it creates a skeleton of the final decomposition by identifying subsystems that resemble established subsystem patterns, such as the body-header pattern and the subgraph dominator pattern [17]. Depending on the pattern used the subsystems are given appropriate names. In the second stage, ACDC completes the decomposition by using an extended version of a technique known as Orphan Adoption [19]. Orphan Adoption is an incremental clustering technique based on the assumption that the existing structure is well established. It attempts to place each newly introduced resource (called an orphan) in the subsystem that seems "more appropriate". This is usually a subsystem that has a larger amount of connectivity to the orphan than any other subsystem.

LIMBO is introduced in [4] as a scalable hierarchical categorical clustering algorithm that builds on the *Information Bottleneck* (*IB*) framework for quantifying the relevant information preserved when clustering. LIMBO has been successfully applied to the software clustering problem [3]. LIMBO's goal is to create clusters whose features contain as much information as possible about the features of their contents. LIMBO considers weights representing dynamic dependencies in the software clustering process.

3. Description of data sets

Static information on a software system represents dependencies between the objects to be clustered. In our case, the objects to be clustered are source files, while the dependencies are procedure calls and variable references. Static information on software systems is categorical, meaning that the objects have attribute values that are taken from a set of discrete values and the values have no specified ordering. We represent static information as a categorical data set by creating an $n \times n$ matrix M, where n is the number of files. Each row of M represents a file i of the software system. The categorical attribute value (CA) in cell (i,j) of M is '0' or '1', where '1' represents that file i calls or references file j and '0' represents that file i does not call or reference file j.

Dynamic information on a software system contains the results of a profiling of the execution of the system, representing how many times each file called procedures in other files during runtime. We represent dynamic information by associating a weight with each CA in the matrix, in the range 0.0–1.0, where 1.0 represents that file i called file j the maximum number of times during the runtime and 0.0 represents that file i did not call file j. Fig. 2 shows an example of a software data set in the form of a matrix.

The weights were derived by normalizing the number of procedure calls during an execution profiling, by dividing all numbers of calls in a column by the maximum number

	file1	file1202
file1	{1,0.33}	0
file2	0	{1,0.5}
file3	{1,0.75}	{1,1.0}

Fig. 2. Cells represent file dependencies. Each cell that has a CA value of '1' is also associated with a weight in the range 0.0–1.0.

of calls in that column. Thus, the weights are real values in the range from 0.0 to 1.0 and there is at least one weight with a value of 1.0 in each column. The rationale behind normalizing the weights this way is that some helper functions get called thousands of times, but we do not want them to have a stronger influence on the clustering process than other important files that get called fewer times.

4. The MULICsoft clustering algorithm

MULICsoft clusters consist of layers, where each layer corresponds to a different value of the similarity criterion used for inserting objects (files) in clusters. An optional final step merges similar clusters into a hierarchy to find more interesting cluster structures.

Each MULICsoft cluster has a *mode*. Assuming that the objects in the data set are described by m categorical attributes, the mode of a cluster c is a vector $\mu_c = \{\mu_{c1}, \ldots, \mu_{cm}\}$. The ith position μ_{ci} is set to '1' if there is at least one object in cluster c that has a value of '1' in the ith attribute. We do not use the most frequent value for each position of the mode as in the traditional k-Modes [8], because with our data set most or all values of the mode would be set to '0'.

MULICsoft ensures that when each object o is clustered it is inserted into the cluster c with the most similar mode μ_c, thus maximizing the similarity between object and mode:

$$similarity(o, \mu_c) \tag{1}$$

where o is an object in the data set and μ_c is the mode of the cluster c in which o is to be inserted. The similarity metric will be described in Section 4.1.

Fig. 3 shows the main part of the MULICsoft clustering algorithm. The algorithm starts by reading all objects from

Input: a set S of objects;

Parameters: (1) $\delta\phi$: the increment for ϕ;
(2) *threshold* for ϕ : the maximum number of values that can differ between an object and the mode of its cluster;

Default parameter values: (1) $\delta\phi = 1$;
(2) *threshold* = the number of categorical attributes m;

Output: a set of clusters;

Method:

1. Insert the first object into a new cluster, use the object as the mode of the cluster, and remove the object from S;

2. Initialize ϕ to 1;

3. Loop through the following until S is empty or $\phi > threshold$

a. For each object o in S

i. Find o's closest cluster c by using the similarity metric to compare o with the modes of all existing cluster(s);

ii. If the number of different values between o and c's mode is larger than ϕ, insert o into a new cluster

iii. Otherwise, insert o into c and update c's mode;

iv. Remove object o from S;

b. For each cluster c, if there is only one object in c, remove c and put the object back in S;

c. If in this iteration no objects were inserted in a cluster with $size > 1$, increment ϕ by $\delta\phi$.

Fig. 3. The MULICsoft clustering algorithm.

the input file and storing them in S. Objects (files) are ordered from lowest to highest degree. The first object is inserted in a new cluster, the object becomes the mode of the cluster and the object is removed from S. Then, it continues iterating over all objects that have not been assigned to clusters yet, to find the closest cluster. In all iterations, the closest cluster for each unclassified object is the cluster with the highest similarity between the cluster's mode and the object, as computed by the similarity metric.

The variable ϕ is maintained to indicate how high the dissimilarity is allowed to be between an object and the closest cluster's mode for the object to be inserted in the cluster. Initially ϕ equals 1, meaning that only one value can differ between an object and the closest cluster's mode. If the number of different values between the object and the closest cluster's mode is greater than ϕ then the object is inserted in a new cluster on its own, else, the object is inserted in the closest cluster and the mode is updated.

At the end of each iteration, all objects assigned to clusters of size one have their clusters removed so that the objects will be re-clustered at the next iteration. This ensures that the clusters that persist through the process are only those containing at least two objects. Objects assigned to clusters of size greater than one are removed from the set of unclassified objects S, so those objects will not be re-clustered.

At the end of each iteration, if no objects have been inserted in clusters of size greater than one, then the variable ϕ is incremented by $\delta\phi$. Thus, at the next iteration the criterion for inserting objects in clusters will be more flexible. The iterative process stops when all objects are classified in clusters of size greater than one, or ϕ exceeds a user-specified *threshold*. If the *threshold* equals its default value of the number of attributes m, the process stops when all objects are assigned to clusters of size greater than one.

The MULICsoft algorithm can eventually classify all objects in clusters, even if the closest cluster to an object is very dissimilar, because ϕ can continue increasing until all objects are classified. Even in the extreme case, where an object o with m attributes has only zero or one value similar to the mode of the closest cluster, it can still be classified when $\phi = m$ or $\phi = m - 1$, respectively.

Fig. 4 illustrates what the results of MULICsoft look like. Each cluster consists of one or more different "layers". The layer of an object represents how high the object's dissimilarity was to the mode of the cluster when the object was assigned to the cluster. The cluster's layer in which an object is inserted depends on the value of ϕ. Bottom layers such as 1000 correspond to higher values of ϕ and have a lower coherence – meaning a higher average dissimilarity between all pairs of objects in the layer. MULICsoft starts by inserting as many objects as possible in top layers – such as layer 1 – and then moves to bottom layers, creating them as ϕ increases.

If an unclassified object has equal similarity to the modes of the two or more closest clusters, then the algorithm tries to resolve this 'tie' by comparing the object to the mode of the top layer of each of these clusters – the top layer of a cluster may be layer 1 or 2 and so on. Each cluster's top layer's mode was stored by MULICsoft when the cluster was created, so it does not need to be recomputed. If the object has equal similarity to the modes of the top layer of all of its closest clusters, the object is assigned to the cluster with the highest bottom layer. If all clusters have the same bottom layer then the object is assigned to the first cluster, since there is insufficient data for selecting the best cluster.

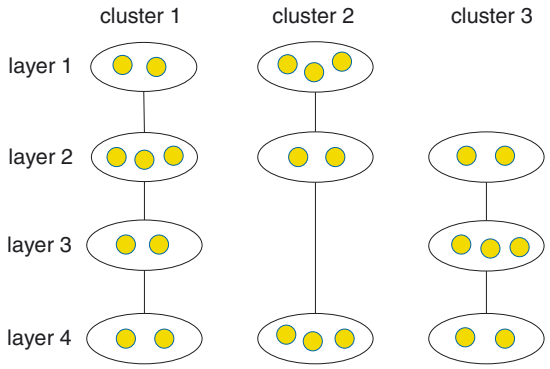

Fig. 4. A MULICsoft cluster consists of one or more layers representing dissimilarities between the objects and mode. Ovals are layers and circles are objects.

4.1. Similarity metric for comparison of objects to modes

A similarity metric is used to find the closest cluster to an object, by computing the similarity between the cluster's mode and the object. MULICsoft handles dynamic information by associating "weights" in the range of 0.0–1.0 with CAs of an object and incorporating these weights in the clustering process through special similarity metrics that consider CAs and weights. The weights were extracted as described in Section 3. We represent the weights of an object o as a vector w_o and the similarity metrics use the weight vectors w_o.

The similarity metric amplifies the weights of the objects as follows:

$$similarity(o,\mu) = \frac{1}{m} \times \sum_{i=1}^{m} \frac{c + w\_o_i^x}{c+1} \times \sigma(o_i,\mu_i)$$

$$\sigma(o_i,\mu_i) = \begin{cases} 1 & \text{if } o_i = \mu_i = 1 \\ 0 & \text{otherwise} \end{cases}$$

The function σ returns 1 if an object o and a mode μ have identical CAs of '1' at a position i and returns 0 otherwise. When calculating the similarity between a mode μ and an object o, pairs of '0' attribute values between mode and object are ignored.

This similarity metric places more importance on high weights (1.0) than low weights (0.0). The parameter c is a constant representing a standard contribution to the similarity metric in case o and μ equal '1' at position i but the weight w_o_i is 0.0. For the experiments presented in this

paper we set $c = 0.25$ as it gives a sufficient contribution to the metric for high quality results; we also tested other values such as $c = 0$ but they gave worse results. The parameter x takes an integer value greater than or equal to 1, which amplifies the relative contributions of the large weight values (close to 1.0) to the similarity. The intuition for this similarity metric is that for each pair of CAs with identical values of '1' between o and μ the contribution to the similarity result should be at least $c/(c+1) = 0.2$ ($c = 0.25$), for the lowest weight of 0.0. The maximum contribution, for the highest weight of 1.0, is 1.0. For example, for $x = 1$ the contribution to the similarity result ranges from 0.28 for a low weight of 0.1 to 1.0 for a high weight of 1.0. For $x = 2$ the contribution to the similarity result ranges from 0.208 for a low weight of 0.1 to 1.0 for a high weight of 1.0.

Fig. 5 shows the shape of the values returned by the similarity metric for $x = 1$ and $x = 3$. Each object in this example has 10 CAs and weights. This graph shows that an object is more likely to be assigned to a cluster if all CAs match the mode with high weights of 1.0. An object is less likely to be assigned to a cluster if all CAs match the mode with lower weights of 0.5 or 0.1 or 0.0. An object is even less likely to be assigned to a cluster if only 1 CA matches the mode with a high or low weight.

4.2. Ordering the objects before clustering

When running MULICsoft with different random orderings of the data set objects (files), the result is often different. The modes and clusters are influenced most by the attribute values of the files that are clustered first in top cluster layers. It makes more sense to cluster first the files of low degree (i.e., files that call few files) and last the files that call many files. Two files of high degree are unlikely to call the exact same files, thus there are unlikely to be many files of high degree in top cluster layers. By ordering the files and presenting them to the clustering process from low to high degree, and by relaxing ϕ gradually, the clusters get an onion-layered structure where files in top layers call similar sets of files and files in bottom layers call less similar sets of files.

4.3. Merging clusters into a tree structure

A hierarchical cluster tree structure can be constructed, by merging pairs of clusters in order of increasing dissimilarity between their modes. Fig. 6 shows the merging process. This process reduces the total number of clusters and may improve the quality of the results. This hierarchical clustering helps an expert to start with 20–30 large clusters and then zoom into more elaborated clusters.

M = upper square matrix of dissimilarities between all pairs of modes.
While (not all clusters are merged into one tree structure)
    d = cell in M with lowest value.
    Merge (link) the pair of clusters corresponding to d.
    Update the mode of the newly merged cluster.
    Update the cells of M with the new dissimilarities.
The dissimilarity between two modes ($\mu_c = \{\mu_{c1}, \cdots, \mu_{cm}\}$ and $\mu_d = \{\mu_{d1}, \cdots, \mu_{dm}\}$) is defined as:

$$dissimilarity(\mu_c, \mu_d) = \sum_{i=1}^{m} \delta(\mu_{ci}, \mu_{di})$$

$$\text{where } \delta(\mu_{ci}, \mu_{di}) = \begin{cases} 1 & \text{if } \mu_{ci} \neq \mu_{di} \\ 0, & \text{if } \mu_{ci} = \mu_{di} \end{cases}$$

Fig. 6. Merging clusters into a hierarchical tree structure.

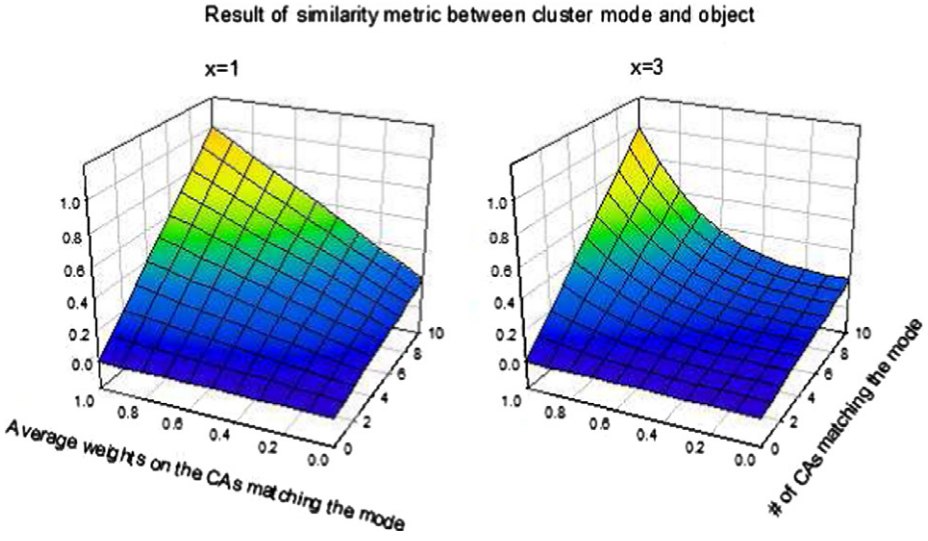

Fig. 5. The function surface of the similarity metric ($c = 0.25$) for $x = 1$ and $x = 3$. There are 10 categorical attributes. Weight values range between 0.0 and 1.0.

4.4. Detection of outliers

MULICsoft will eventually put all the objects in clusters if the *threshold* for ϕ equals its default value of the number of attributes m. When ϕ equals m, any object that remains unclassified will be inserted in the lowest layer of a cluster. This is undesirable if the object is an outlier and has little similarity with any cluster. The user can disallow this situation from happening by specifying a value for *threshold* that is less than m. In this case when ϕ exceeds the maximum allowed value specified by *threshold*, any remaining objects are treated as outliers. We showed that top layers are more reliable than lower layers in [2].

4.5. MULICsoft characteristics for software clustering

MULICsoft includes characteristics specific for software clustering, allowing the incorporation of both static and dynamic system information in the clustering process. Such characteristics include the mode's updating and special similarity metrics used to compute the similarity between a mode and an object.

While the MULICsoft clustering algorithm follows the basic framework of k-Modes [8], it has substantially different characteristics. *First*, clusters are layered. *Second*, the number of clusters is not specified by the user – clusters are created, removed or merged, as the need arises. k-Modes requires the user to specify the number of clusters and the algorithm builds and refines the specified number of clusters. *Third*, all MULICsoft clusters are of size two or greater.

Some of the benefits of our clustering approach include: *a*. Multi-layer clustering of a software system may help the user identify files making similar calls, as well as file calls that occur most frequently within each cluster. The multi-layer clustering can help experts find top-layer files in a cluster that make the most frequent calls. *b*. The hierarchical cluster organization allows the expert to start with 20–30 large clusters and then zoom into clusters to study smaller subclusters. The number of clusters can be reduced or increased as the user wishes. *c*. Each cluster can have a label associated with it, based on the most frequently called files in the cluster. *d*. The multi-layer clustering can help visualization tools represent the software architecture better.

5. Results for clustering mozilla with MULICsoft

In order to evaluate the applicability of MULICsoft to the software clustering problem, we applied it to the Mozilla software system and compared its output to that of other well-established software clustering algorithms. We experimented with Mozilla version 1.3 that was released in March 2003. It contains approximately four million lines of C and C++ source code. We have placed all of our results online; we have also made available the complete list of the 1202 Mozilla files used in our clustering experiments, as well as the static and dynamic information, thus allowing replication of all our experiments.[1]

A static dependency graph was extracted using Swagkit [16]. In order to extract dynamic information one needs a comprehensive test suite that exercises all features of the given software system. For most software systems, such a test suite does not exist. In our case, we were able to use the smoke tests provided by the project combined with test suites published for individual subsystems. This yielded a dynamic dependency graph comprised of 1202 source files.

An authoritative decomposition of the Mozilla source files for version M9 was presented in Ref. [7]. For the evaluation portion of our work, we used an updated authoritative decomposition for version 1.3 [21]. The authoritative decomposition consists of 10 clusters.

We compared MULICsoft to the three software clustering algorithms presented in Section 2: ACDC [17], Bunch [12], LIMBO [4]. All algorithms were given information concerning only the 1202 source files present in the dynamic dependency graph.

To evaluate the clustering results we compared them with the authoritative manual decomposition, using the MoJo distance measure[2] [17,18]. MoJo measures the distance between two decompositions of the same software system by computing the number of Move and Join operations one needs to perform in order to transform one to the other. Intuitively, the smaller the distance of a proposed decomposition to the authoritative one, the more effective the algorithm that produced it.

MULICsoft clusters the 1202 Mozilla files into 100–200 clusters without merging the clusters after the clustering process. The clusters produced for Mozilla before merging have sizes ranging from 3 to 37 files. The results indicate that MULICsoft outperforms other software clustering algorithms, such as LIMBO, Bunch and ACDC. The MoJo distances for ACDC, Bunch and LIMBO applied to clustering the Mozilla software system are shown in Table 1. MULICsoft clusters all Mozilla system files, without treating any as outliers, giving MoJo distances as low as 377, as explained in the next section.

5.1. Results for various x values of the similarity metric

Table 2 shows results for various values of x. The experiments use a linear growth of ϕ by setting ϕ to an initial value of 1 and increasing it by a constant value $\delta\phi$, after each loop where no object was classified in a cluster of size greater than one. We set *threshold* equal to its default value of the number of attributes m, so that no objects are treated as outliers and all 1202 files are clustered. We do not merge the clusters after the clustering process.

As Table 2 shows, the MoJo distance to the authoritative manual decomposition is significantly lower than the

[1] http://www.cs.yorku.ca/~billa/MULICsoftware05/

[2] A Java implementation of MoJo is available for download at: http://www.cs.yorku.ca/~bil/downloads/

Table 1
ACDC, Bunch, LIMBO results for clustering Mozilla

Software clustering algorithm	MoJo distance	Number of clusters	Files classified
ACDC	439	205	1202
Bunch	440	21	1202
LIMBO	438	75	1202

Table 2
MULICsoft results for clustering Mozilla with different similarity metric x values

Similarity metric	MoJo distance	Number of clusters	$\delta\phi$
$x = 1$	391	204	150
$x = 2$	383	197	150
$x = 3$	382	198	150
$x = 4$	382	193	150
$x = \mathbf{5}$	**377**	**194**	**150**

distances of ACDC, Bunch and LIMBO. For a value of $x = 5$, the results are especially good with a MoJo distance of 377 to the authoritative decomposition. The reason why $x = 5$ produces the best results is that the similarity metric amplifies significantly the effect of the high weights on the clustering process. We tried setting x to even higher values, such as 6, 7 and 9, but the MoJo distance no longer decreased. Thus, a high value for the parameter x improves the results until a specific point. As x decreases to 2 and 1, the results are still good, with MoJo distances of 383 and 391, respectively.

For the experiments presented in this paper we set $c = 0.25$. We also experimented with setting $c = 0$. The best resulting MoJo distance for $c = 0$ was 417 by using a lower value of $\delta\phi$ of 80. The reason for this is that with $c = 0.25$ more files are classified in the correct cluster during the first and second iterations, because of the amplified effect of the weights on the clustering process. With $c = 0$, on the other hand, fewer files are classified correctly during the first and second iterations and the lower value of $\delta\phi$ allows some of the files to be considered instead at the next iterations.

5.2. MULICsoft with linear and exponential growth of ϕ

We experimented with a linear growth of ϕ by setting ϕ to an initial value of 1 and increasing it by a *constant* value $\delta\phi$ after each loop at which no object was placed in a cluster of size greater than one. Fig. 7 shows our results for a linear growth of ϕ by constant value $\delta\phi$. We let *threshold* have its default value equal to the number of attributes m, so that no objects are treated as outliers and all 1202 files are clustered. We do not merge the clusters after the clustering process.

As Fig. 7 shows, a value of $\delta\phi$ between 70 and 150 gives the best results overall, with MoJo distances as low as 377. The reason why a high value of $\delta\phi$ is used is that sufficient files should be clustered at each iteration so that the modes of the clusters are given the opportunity to change, as opposed to remaining static.

For both $x = 5$ and $x = 1$ a value of $\delta\phi = 150$ gives the best results with MoJo distances of 377 and 391, respectively. For $x = 5$ the best result has 194 clusters, while for $x = 1$ the best result has 204 clusters. The reason for this is that with $x = 5$ more files are classified in the correct cluster during the first and second iterations, because of the amplified effect of the weights on the clustering process. With $x = 1$, on the other hand, fewer files are classified correctly during the first and second iterations and some of the files are considered again at the next iterations, resulting in more clusters.

We also experimented with increasing ϕ exponentially (2^y), by setting ϕ to an initial value of 1 and multiplying it by 2 after each loop at which no object was classified in a cluster of size greater than one. The MoJo distance increases to 456 and the number of clusters $k = 280$. Even though the exponential growth of ϕ does not produce the best results in this case, it can still produce good results when we treat some objects as outliers, as described in Section 5.4.

5.3. Results after merging clusters into a tree structure

MULICsoft provides the capability to merge clusters into a hierarchical tree structure, as described in Section

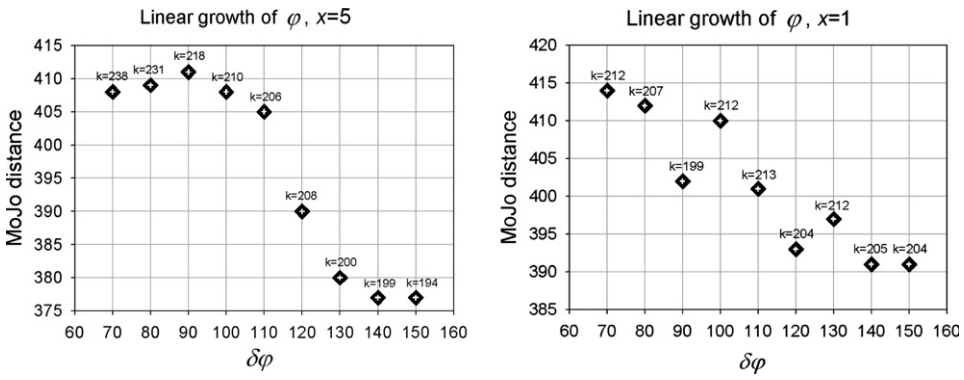

Fig. 7. MULICsoft results for clustering Mozilla with $x = 5$ and $x = 1$, respectively. The initial value of ϕ is 1 and ϕ increases linearly by constant value $\delta\phi$. k is the number of clusters.

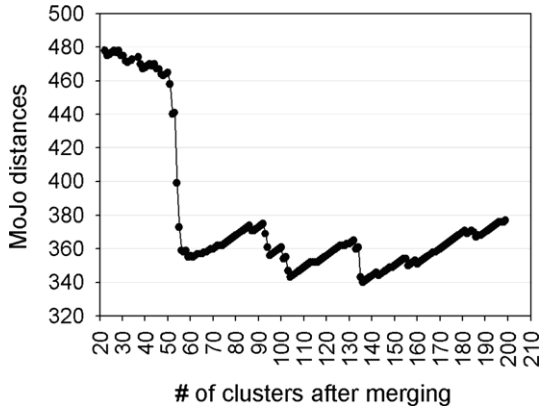

Fig. 8. MoJo distances after merging down to various numbers of clusters.

4.3. Fig. 8 presents the MoJo distances after merging down to various numbers of clusters. Without any merging the number of clusters is 199. A value of $x = 5$ is used. The initial value of ϕ is 1 and it increases linearly by constant value $\delta\phi = 140$. We let *threshold* have its default value equal to the number of attributes m, so that no objects are treated as outliers and all 1202 files are clustered.

As Fig. 8 shows, we merge clusters until the number of clusters decreases to 20. The MoJo distance to the authoritative decomposition decreases overall until there are 50 merged clusters in total. At the best point, at 136 merged clusters the MoJo distance is 340. As clusters decrease below 50, the MoJo distance starts increasing. An expert could start by investigating the 40–50 large clusters and then zoom in to clusters to explore their refined structure and their subclusters.

5.4. Treating objects as outliers by setting a threshold for ϕ

Some times it may be desirable to treat the objects in bottom layers of clusters as outliers. Objects are treated as outliers by setting the *threshold* for ϕ to a value less than the number of attributes m, as discussed in Section 4.4. When ϕ exceeds the maximum allowed value specified by *threshold*, any remaining objects are treated as outliers by classifying them independently in clusters of size one. For example, setting the *threshold* for ϕ to the value 150 means that clustering will stop at layer 150 and any objects that would be clustered in layers greater than 150 are treated as outliers. We showed that lower layers are less reliable than higher layers in [2]. We experiment with various *thresholds* for ϕ, for both linear and exponential growths of ϕ. We use a value of $x = 2$. We do not merge the clusters after the clustering process.

Table 3 shows the results for placing all outliers in one cluster together. We also experimented with placing each outlier in an independent cluster of size one; in this case many Moves and Joins need to be performed for the computed decomposition to reach the authoritative manual decomposition and the MoJo distance is high. The MoJo distance with all outliers placed in one cluster together is significantly lower than if each outlier is placed in an independent cluster of size one.

Table 3
MULICsoft results for setting a *threshold* for ϕ and treating some files as outliers

$\delta\phi$	*Threshold* for ϕ	MoJo dist.	Number of clusters	Number of outliers
Linear growth of ϕ				
130	131	397	213	83
120	121	400	208	93
10	50	500	305	230
10	80	484	330	96
50	51	469	260	191
60	61	449	250	155
99	100	420	215	104
Exponential growth of ϕ				
Multiply ϕ by 2	32	540	318	285
Multiply ϕ by 2	64	515	345	135

All outliers are placed in one cluster. The initial value of ϕ is 1. The similarity metric is used with $x = 2$.

The MoJo distance decreases further if the outliers are ignored and the distance is computed between the intersection of files in the computed decomposition with files in the authoritative decomposition. A distance metric other than MoJo could be useful for measuring the distance between a computed decomposition and an authoritative decomposition when outliers are involved.

5.5. Discussion: structure and consistency of decompositions

Experts want an automated tool for clustering software systems, such that a human can inspect the software architecture in a more efficient manner. While reverse engineering, an expert is interested in knowing which calls occur frequently and uniquely within each cluster and are most characteristic of the cluster. Our proposed software clustering approach is based on calling relationships and creates clusters that have a multi-layered structure and are hierarchically organized. The clusters have an onion-layered structure based on level of similarity of the files to the cluster. The hierarchical structure gives the expert a semantic hierarchy for software inspection.

Most clusters have several layers, such as 1, 141 and 281 (for $\delta\phi = 140$). Several files are always clustered in top layers, regardless of the value of $\delta\phi$. Since the files are ordered from low to high degree as described in Section 4.2, the files in top layers are usually calling files that are very characteristic of the cluster, meaning that within the entire system they are frequently called by files in the cluster. Fig. 9 illustrates an example of a cluster containing *JavaScript* files, which in the authoritative decomposition are all placed in one cluster together. Files in top layer 1 call files characteristic of the cluster, while files in bottom layer 141 call other files too. For instance, in this cluster the files in layer 1 call

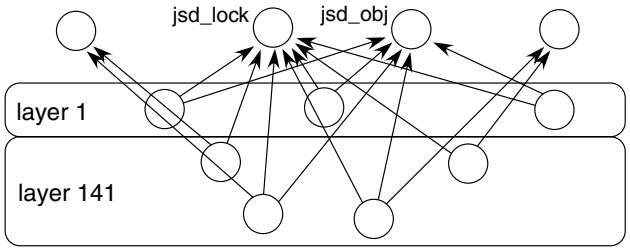

Fig. 9. A MULICsoft cluster of *JavaScript* files with two layers: 1 and 141. Circles represent files and arrows represent calls. The files in layer 1 call the `jsd_ock.c` and `jsd_obj.c` files, while the files in layer 141 call other files too.

the `jsd_lock.c` and `jsd_obj.c` files which are JavaScript specific.

We notice that there are groups of files always clustered together. For example, the JavaScript files are usually clustered together in the same clusters. Without merging clusters, the JavaScript files are spread out over about 10 clusters, which are pretty homogeneous containing mostly JavaScript files. After merging down to a total number of 50 clusters, the JavaScript files are spread out over 2–3 merged clusters only. As expected, the merged clusters contain a higher absolute number of non-JavaScript files than the non-merged clusters.

6. Inputting additional categorical data

We integrated the following categorical data sets with the Mozilla file data set, to produce improved results when MULICsoft clustering is applied to the integrated data sets.

- Developers (Dev): The ownership information, i.e., the names of the developers involved in the implementation of the file.
- Directory path (Dir): The full directory path for each file. In order to increase the similarity of files residing in similar directory paths, the set of all subpaths for each path is included.
- Lines of code (Loc): The number of lines of code for each of the files.The values are discretized by dividing the full range of loc values into the intervals {0; 100}, {100; 200}, {200; 300}, etc. Each file is given a feature such as RANGE1, RANGE2, RANGE3, etc.
- Time of last update (Tim): The time-stamp of each file on the disk.Only the month and year are included.

Table 4 shows the MULICsoft MoJo distances to the authoritative Mozilla decomposition with inputting additional categorical data sets. The MoJo distances to the authoritative decomposition improve significantly with inputting additional categorical data sets to MULICsoft. After inputting all four additional data sets of Dev + Dir + LocEQ + Tim, the result is 228, which is much better than the best previous result of 388 for not inputting any additional categorical data. As fewer additional categorical data sets are input, the MoJo distance decreases further. For three additional categorical data sets the distance decreases to 192 and 210. For two additional categorical data sets the distance decreases to 211.

Table 4
MULICsoft results for clustering Mozilla with additional categorical data

Categorical data sets	MoJo distance	Number of clusters
Dev + Dir + LocEQ + Tim	228	170
Dev + Dir + LocEQ	210	168
Dev + Dir + Tim	192	175
Dev + Dir	211	165

The initial value of ϕ is 1 and it increases linearly by constant value $\delta\phi = 80$. The similarity metric is used with $x = 3$ *threshold* has its default value. No merging is done on the clusters after the clustering process.

7. Using an alternative evaluation measure

The evaluation results presented in the previous sections were based on the MoJo distance between the decomposition created by MULICsoft and the authoritative decomposition of Mozilla. In order to avoid basing our results solely on the use of MoJo distance, we measured the similarity of the candidate decompositions prepared by MULICsoft to the authoritative one using also the Koschke–Eisenbarth (KE) measure [10].

The KE measure is loosely based on the amount of overlap between corresponding clusters in the two decompositions. Its value is normalized to a percentage scale. The higher it is, the closer the two compared decompositions are. As a result, if the results obtained from the two evaluation methods (MoJo and KE) are to be congruent, we would expect that as the value of KE increases, MoJo distance decreases, and vice versa.

We applied both evaluation methods to 115 different candidate decompositions created by MULICsoft. These decompositions were created using a variety of thresholds, as well as many different combinations of additional information. The relation between the KE and MoJo values obtained are presented in Fig. 10.

A first observation would be that the two measures do not appear to be congruent, since the scatterplot does not seem to be an approximation of a monotonously decreasing function as expected. However, closer inspection shows that there are two clusters of results. The first one, in the top left part of the graph, corresponds to decompositions of large MoJo distance and low KE value, while the second one, in the bottom right part of the graph, corresponds to smaller MoJo distance and higher KE values. This indicates that both measures clearly differentiate effective decompositions from not so effective ones, a result that agrees with the experimental data presented in Ref. [3].

These results indicate that, while small differences in MoJo distance are not indicative of a clear ranking between two candidate decompositions, larger MoJo

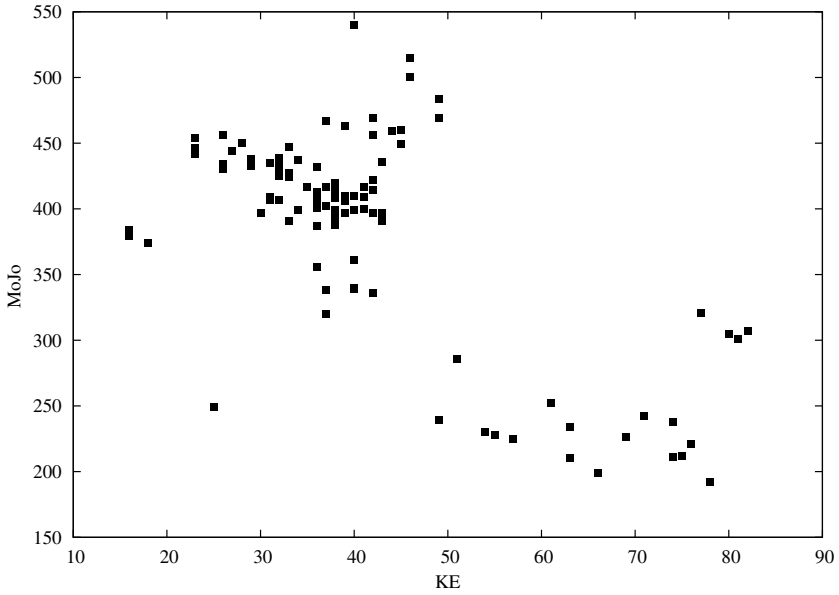

Fig. 10. The relation between KE and MoJo values for 115 different decompositions.

distance discrepancies, as was the case in many of the experimental results presented in this paper, can clearly designate effective candidate decompositions. Therefore, we believe that the results presented in earlier sections are mostly independent of the evaluation method used.

8. Computational complexity

The best-case complexity of MULICsoft has a lower bound of $\Omega(mNk)$ and its worst-case complexity has an upper bound of $O(mN^2 \frac{threshold}{\delta\phi})$. Often $m \ll N$, since in a categorical data set the number of attributes m is usually smaller than the number of objects N. The cost is related to the number of clusters k generated throughout the process. Usually, the number of clusters k is smaller than the number of objects N and all objects are clustered in the initial iterations, thus N often dominates the cost. The worst-case runtime would occur for the rather unusual data set where all objects were extremely dissimilar to one another, such that the algorithm had to go through all m iterations and all N objects were clustered in the last iteration when $\phi = \mathrm{m}$. Decreasing the value of *threshold* or increasing the value of $\delta\phi$ improves the runtime, often without weakening the clustering quality. A value of $\delta\phi$ greater than 1 often improves the clustering quality by allowing the modes to change from one iteration to another [2]. The MULICsoft complexity is comparable to that of k-Modes of $O(mNkt)$, where t is the number of iterations [8].

8.1. Runtime evaluation

Our experiments were performed on a Sun Ultra 60 with 256 MB of memory and a 300 MHz processor. Table 5 shows the run times it took for MULICsoft to cluster the files of the Mozilla system. Most of our trials had runtimes of less than 30 s.

Table 5
MULICsoft runtimes, in seconds

Time (seconds)	Sim. metric	$\delta\phi$	*Threshold*
12	$x = 3$	90	Default (1202)
12	$x = 3$	120	Default (1202)
29	$x = 2$	10	50
13	$x = 2$	120	121
13	$x = 3$	50	Default (1202)

The initial value of ϕ is 1 and it increases linearly by constant value $\delta\phi$.

9. Conclusion and future work

We have presented the MULICsoft software clustering algorithm. MULICsoft creates decompositions that are close to manually created ones. MULICsoft does not sacrifice the quality of the results for the number of clusters, which in k-Modes is defined strictly before the process [2].

For each cluster, MULICsoft forms layers of varying interdependencies between the files. This can be useful for software comprehension. MULICsoft starts by forming a first layer of highly interdependent files, using strict criteria concerning which files to insert in the layer. The first layer of a cluster contains the highly interdependent files that are at the core of a subsystem. As the process continues, MULICsoft relaxes its criteria, forming layers with files that are less interdependent than the previous layers. The multi-layer structure of MULICsoft is ideal for clustering software system data. MULICsoft clusters are representative of the underlying patterns in a software system, because differing layers of interdependencies exist in a cluster between files.

MULICsoft similarity metrics consider dynamic system information by incorporating weights on the file interdependencies that are derived from a runtime profiling of the system. In the end, the human expert has the option of merging clusters that are very similar to build larger clusters and reduce the number of clusters.

We evaluated the quality of MULICsoft results on the Mozilla software system by computing the MoJo distance to an existing expert-defined authoritative system decomposition. On this data set, the MULICsoft MoJo distance to the authoritative decomposition was lower than the distances of LIMBO [4], Bunch [12] and ACDC [17]. The MULICsoft results are improved by inputting additional categorical data sets or ordering the objects by the frequency of their categorical attribute values. Our evaluation of the quality of MULICsoft results on Mozilla using the alternative KE measure supports the high quality of the results. Finally, we showed that the runtime of MULICsoft was satisfactory as it took between 10 and 30 s.

The results for identifying outliers indicate that the distance to the authoritative decomposition increases if in the computed decomposition each outlier is inserted in an independent cluster of size one. We experimented with different ways to handle outliers, such as inserting them all in one cluster. This decreases the MoJo distance, since fewer Moves and Joins need to be performed in the computed decomposition to reach the authoritative manual decomposition. Future work will include designing a different distance measure for computing the distance between a computed decomposition and an authoritative decomposition, when outliers are involved.

One direction worth pursuing in the future is to experiment with MULICsoft on more cases, both from open-source systems and from traditional industrial systems. Another direction worth pursuing is to improve the method for merging clusters that are similar, to build larger clusters, after the clustering process. This will hopefully produce better MoJo distance results than the current merging.

References

[2] B. Andreopoulos, A. An, X. Wang, MULIC: Multi-layer Increasing Coherence Clustering of Categorical Data Sets, Technical Report # CS-2004-07, Department of Computer Science and Engineering, York University, 2004.

[3] P. Andritsos, V. Tzerpos, Information-theoretic software clustering, IEEE Transactions on Software Engineering 31 (2) (2005).

[4] P. Andritsos, P. Tsaparas, R.J. Miller, K.C. Sevcik, LIMBO: scalable clustering of categorical data, in: Proceedings of the Ninth International Conference on Extending DataBase Technology (EDBT'04), March 2004.

[5] L.A. Belady, C.J. Evangelisti, System partitioning and its measure, Journal of Systems and Software 2 (1981) 23–29.

[6] S.C. Choi, W. Scacchi, Extracting and restructuring the design of large systems, IEEE Software (1990) 66–71.

[7] W. Godfrey, E.H.S. Lee, Secrets from the Onster: extracting Mozilla's software architecture, in: Proceedings of the Second International Symposium on Constructing Software Engineering Tools (CoSET), 2000.

[8] Z. Huang, Extensions to the k-Means algorithm for clustering large data sets with categorical values, Data Mining and Knowledge Discovery 2 (3) (1998) 283–304.

[9] D.H. Hutchens, V.R. Basili, system structure analysis: clustering with data bindings, IEEE Transactions on Software Engineering 11 (8) (1985) 749–757.

[10] R. Koschke, T. Eisenbarth, A framework for experimental evaluation of clustering techniques, in: Proceedings of the Eighth International Workshop on Program Comprehension, June 2000, pp. 201–210.

[11] C. Lindig, G. Snelting, Assessing modular structure of legacy code based on mathematical concept analysis, in: Proceedings of the 19th International Conference on Software Engineering, May 1997, pp. 349–359.

[12] S. Mancoridis, B.S. Mitchell, Y. Chen, E.R. Gansner, Bunch: a clustering tool for the recovery and maintenance of software system structures, in: Proceedings of the International Conference on Software Engineering (ICSM'99), 1999.

[13] B.S. Mitchell, S. Mancoridis, Comparing the decompositions produced by software clustering algorithms using similarity measurements, in: Proceedings of the International Conference on Software Maintenance (ICSM'01), 2001, pp. 744–753.

[14] H.A. Müller, M.A. Orgun, S.R. Tilley, J.S. Uhl, A reverse engineering approach to subsystem structure identification, Journal of Software Maintenance: Research and Practice 5 (1993) 181–204.

[15] R.W. Schwanke, An intelligent tool for re-engineering software modularity, in Proceedings of the 13th International Conference on Software Engineering, May 1991, pp. 83–92.

[16] http://www.swat.uwaterloo.ca/~swagkit.

[17] V. Tzerpos, R.C. Holt, ACDC: an algorithm for comprehension-driven clustering, in: Proceedings of the Seventh Working Conference on Reverse Engineering (WCRE'00), 2000, pp. 258–267.

[18] V. Tzerpos, R.C. Holt, MoJo: a distance metric for software clusterings, in: Proceedings of the Sixth Working Conference on Reverse Engineering (WCRE'99), 1999, pp. 187–196.

[19] V. Tzerpos, R.C. Holt, The orphan adoption problem in architecture maintenance, in: Proceedings of the Fourth Working Conference on Reverse Engineering (WCRE'97), Amsterdam, October 1997, pp. 76–82.

[20] T.A. Wiggerts, Using clustering algorithms in legacy systems remodularization, in: Proceedings of the Fourth Working Conference on Reverse Engineering (WCRE'97), October 1997, IEEE Computer Society Press, pp. 33–43.

[21] C. Xind, V. Tzerpos, Software clustering based on dynamic dependencies, in: Proceedings of the Ninth European Conference on Software Maintenance and Reengineering (CSMR'05), Manchester, March 2005, pp. 124–133.