Multiple Layer Clustering of Large Software Systems

Bill Andreopoulos
Department of Computer Science and Engineering, York University, 4700 Keele Street, Toronto, M3J1P3
billa@cs.yorku.ca

Aijun An
Department of Computer Science and Engineering, York University, 4700 Keele Street, Toronto, M3J1P3
aan@cs.yorku.ca

Vassilios Tzerpos
Department of Computer Science and Engineering, York University, 4700 Keele Street, Toronto, M3J1P3
bil@cs.yorku.ca

Xiaogang Wang
Department of Mathematics and Statistics, York University, 4700 Keele Street, Toronto, M3J1P3
stevenw@mathstat.yorku.ca

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.

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 significantly facilitates 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 [9, 8, 7]. 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[1] 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 the run time) 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 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

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

consists of layers formed gradually through iterations, by reducing the similarity criterion for inserting objects in layers of a cluster at different iterations.

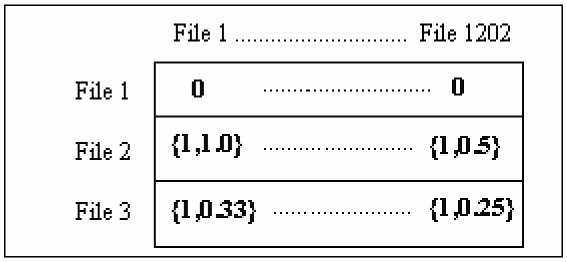
Section 2 describes the Mozilla data set used for clustering. Section 3 gives an overview of previous software clustering tools. Section 4 describes the MULICsoft clustering algorithm. Section 5 describes the experimental results on the Mozilla system. Section 6 discusses inputting additional data to MULICsoft. Section 7 discusses the runtime performance. Section 8 concludes the paper and discusses future work.

2. Description of Data Sets

Static information on a software system represents dependencies between the objects to be clustered. 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, where N is the number of files. Each row of the matrix represents a file i of the software system, along with the files that i may call or reference during execution. The categorical attribute value (CA) in cell *(i,j)* of the matrix is 'zero' or 'one', where 'one' represents that file i calls or references file j and 'zero' 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 the run time. Each row of the data set represents a file x of the software system, along with the files that x called as well as how many times x called them during the profiled run time. We represent dynamic information by associating a weight with each CA in the matrix, in the range 0.0 to 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. Figure 1 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 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.



	File 1	....	File 1202
File 1	0	....	0
File 2	{1,1.0}	....	{1,0.5}
File 3	{1,0.33}	....	{1,0.25}

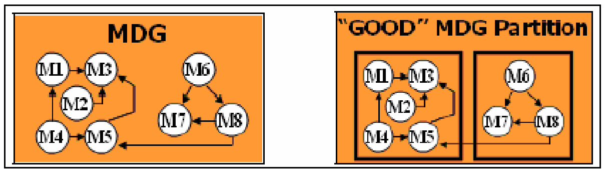
Figure 1. Cells representing file dependencies have values {CA: zero or one, weight: 0.0-1.0}.

We have applied the MULICsoft algorithm to cluster the Mozilla software system. The process of extracting static and dynamic information for Mozilla was presented in [3]. There are 1202 objects in the Mozilla data set, corresponding to 1202 source files of the Mozilla system. We use both categorical data and weights in clustering. The CAs are boolean values of 'zero' or 'one' describing the dependencies between the Mozilla files. A weight in the range 0.0 to 1.0 is attached to each CA to indicate how strongly the corresponding CA should influence the clustering process.

3. Related Work

Several clustering algorithms for software have been presented in the literature [8, 9, 3]. Some of the previous software clustering tools can consider the dynamic information (i.e., the number of function calls during the run time) in the clustering process [12, 3] but most can not. In this section, we describe three established clustering algorithms: BUNCH, ACDC and LIMBO.

Bunch is a clustering tool intended to aid the software developer and maintainer in understanding, verifying and maintaining a source code base [8]. The input to Bunch is a Module Dependency Graph (MDG). Figure 2 shows an MDG graph. Bunch views the clustering problem as trying to find a good partition of an MDG graph. Bunch views 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. Figure 2b shows a "good" partitioning of Figure 2a. 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) [8].



Figures 2a,b. An MDG graph, from [8].

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 [9]. 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 [11]. 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.

4. The MULICsoft Clustering Algorithm

MULICsoft is an extension of the k-Modes clustering algorithm for categorical data sets [6]. The k-Modes clustering algorithm requires the user to specify the number of clusters to be produced and the algorithm builds and refines the specified number of clusters. Each cluster has a mode associated with it. Assuming that the objects in the data set are described by m categorical attributes, the mode of a cluster is a vector $Q=\{q_1, q_2, \ldots, q_m\}$ where q_i is the most frequent value for the ith attribute in the given cluster.

MULICsoft makes substantial changes to the k-Modes algorithm. The purpose of the MULICsoft clustering algorithm is to maximize the similarity between the object and the mode of the cluster in which the object is inserted:

$$similarity(o_i, \text{mode}_i) \qquad (1)$$

where o_i is the ith object in the data set and mode_i is the mode of the ith object's cluster. There are various options for the similarity metric that will be described in Section 4.4. Maximizing formula (1) ensures that all objects are as similar to their clusters' modes as possible, when the objects are clustered.

The MULICsoft algorithm has the following characteristics. First, the number of clusters is not specified by the user. Clusters are created, removed or merged during the clustering process, as the need arises. Second, it is possible for all objects to be assigned to clusters of size two or greater by the end of the process. However, outliers are assigned to separate clusters of size one. Third, clusters are layered.

```
Input:  (1)  a set $S$ of objects;
Parameters:  (1)  $\delta\varphi$ : the increment for $\varphi$;
             (2)  *threshold* for $\varphi$ : the maximum number
                  of values that can differ between an
                  object and the mode of its cluster;
Default parameter values: (1)  $\delta\varphi = 1$;
                          (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 $\varphi$ to 1;
  3. Loop through the following until S is empty or $\varphi$ is
     greater than the specified *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 $\varphi$, 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 loop no objects were placed in a
        cluster with size > 1, increment $\varphi$ by $\delta\varphi$.
```

Figure 3. The MULICsoft clustering algorithm.

Figure 3 shows the main part of the MULICsoft clustering algorithm. The algorithm starts by reading all objects from the input file and storing them in *S*. 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 strong the similarity has 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 the similarity has to be very strong 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 2 objects for which the required similarity can be found. Objects assigned to clusters with 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 $\delta\varphi$. 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 not that similar, because φ can continue increasing until all objects are classified. Even in the extreme cases, 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 $\varphi = m$ *or* $\varphi = m-1$, respectively.

Figure 4 illustrates what the results of MULICsoft look like. Each cluster consists of many different "layers" of objects. The layer of an object represents how strong the object's similarity 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. Lower layers have a lower coherence - meaning a lower average similarity between all pairs of objects in the layer - and correspond to higher values of φ. MULICsoft starts by inserting as many objects as possible in top layers – such as layer 1 - and then moves to lower layers, creating them as φ increases.

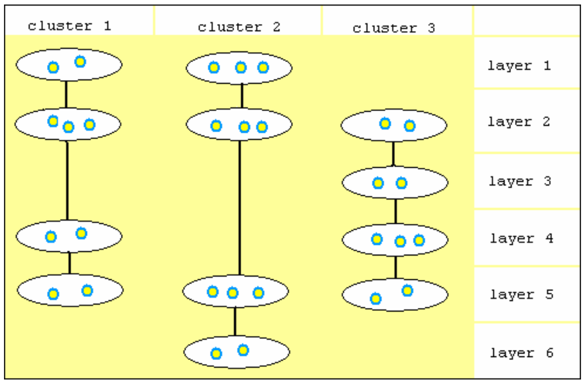


Figure 4. MULICsoft results. Each cluster consists of one or more different layers representing different similarities of the objects attached to the cluster.

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.

The complexity of MULICsoft is $O(N^2)$, where *N* is the number of objects. Most of our trials had a runtime of less than 30 seconds. Increasing $\delta\varphi$ or decreasing *threshold* reduces the runtime, often without hurting the quality of the results [2].

4.1. Merging of clusters

We should generally avoid the situation where the similarity of the top layers of two different clusters is stronger than the similarity of the top and bottom layer of the same cluster. To avoid this, after the clustering process MULICsoft can merge pairs of clusters whose top layers' modes' dissimilarity is less than the maximum layer depth of the two clusters. For this purpose, MULICsoft preserves the modes of the top layers of all clusters. This process reduces the total number of clusters and may improve the quality of the results. This process is described as follows:

```
for (c = first cluster to last cluster)
    for (d = c+1 to last cluster)
        if the dissimilarity between c's mode and d's
          mode is less than the maximum layer
          depth of c and d, merge c into d and break
          the inner loop;
```

where the dissimilarity between two modes ($Q_c = \{q_{c1}, ..., q_{cm}\}$ and $Q_d = \{q_{d1}, ..., q_{dm}\}$) is defined as:

$$\text{dissimilarity}(Q_c, Q_d) = \sum_{i=1}^{m} \delta(q_{ci}, q_{di})$$

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

4.2. Dealing with 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 by classifying each object in a separate cluster of size one. We showed that top layers are more reliable than lower layers in [2].

4.3. 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.

All categorical attribute values (CAs) of an object have "weights" in the range of 0.0 to 1.0 associated with them, which represent dynamic information derived from profiling the execution of a system. The weights were extracted as described in Section 2. We represent the weights of an object *o* as a vector *w_o*.

A position of the mode of a cluster is set to 'one' if there is at least one object in the cluster that has a CA of 'one' in the corresponding position, or has a weight greater than 0.0 at the corresponding position. We do not use the most frequent value for each position of the mode, because with our software data set most or all values of the mode would be set to 'zero'.

When calculating the similarity between a mode and an object, pairs of 'zero' attribute values between mode and object are ignored.

Besides storing the boolean values of 'zero' or 'one' for the mode μ of a cluster, we also store real numbers for each position of the mode, which represent the sum of all weights at that position over all objects allocated to the cluster. We represent this special mode as w_μ.

Special similarity metrics are used to compute the similarity between a mode and an object.

4.4. Similarity metrics 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 with CAs and incorporating these weights in the clustering process through special similarity metrics that consider CAs and weights. The similarity metrics use the weight vectors *w_o* and/or w_μ. The function σ returns 1 if an object *o* and a mode μ have identical CAs of 'one' at a position, and returns 0 otherwise:

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

Similarity metric 1. The first similarity metric uses both the weights of the objects and the mode:

$$\textit{similarity}(o, \mu) = \sum_{i=1}^{m} w\_o_i \times w\_\mu_i \times \sigma(o_i, \mu_i)$$

Similarity metric 2. The second similarity metric uses only the weights of the objects:

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

Similarity metric 3. The third similarity metric amplifies the weights of the objects as follows:

$$\textit{similarity}(o, \mu) = \sum_{i=1}^{m} \frac{x - (4 \times w\_o_i)}{5 - (4 \times w\_o_i)} \times \sigma(o_i, \mu_i)$$

The parameter *x* takes an integer value greater than 5. This similarity metric places more importance on high weights (1.0) than low weights (0.0). The intuition for this formula is that for each pair of CAs with identical values of 'one' between *o* and μ the contribution to the similarity result should be at least 1.0, for the lowest weight of 0.0. The maximum contribution, for the highest weight of 1.0, depends on the integer value of *x*. For example, for x=6 the contribution to the similarity result ranges from 1.2 for a low weight of 0.1 to 2.0 for a high weight of 1.0. For x=9 the contribution to the similarity result ranges from 1.8 for a low weight of 0.1 to 5.0 for a high weight of 1.0.

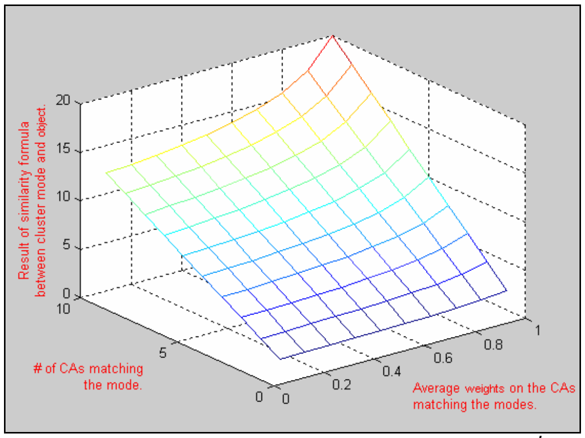


Figure 5. The function surface of the 3rd *similarity metric*, for *x=6* and weight values between 0.0 and 1.0.

Figure 5 shows the shape of the values returned by the 3rd similarity metric for *x=6*. 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, than if all CAs match the mode with medium weights of 0.5, than if all CAs match the mode with low weights of 0.1, than if 1 CA matches the mode with a high weight, than if 1 CA matches the mode with a low weight.

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 built Mozilla under Linux and extracted its static dependency graph using CPPX and a dynamic dependency graph using jprof. A decomposition of the Mozilla source files for version M9 was presented in [5]. For the evaluation portion of our work, we used an updated authoritative decomposition for version 1.3 [12]. We have placed all of our detailed results online[2].

We compared MULICsoft to the following software clustering algorithms: ACDC [9], BUNCH [8], LIMBO [4].

To evaluate the clustering results we compared them with the authoritative manual decomposition, using the MoJo distance measure[3] [10, 9]. 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 such as 388, 397, 399, 424, as explained in the next section.

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

5.1. Results for different similarity metrics

Table 2 shows the results for each of the 3 similarity formulas. The experiments use a linear increase of φ by setting it to an initial value of 1 and increasing it by a constant value $\delta\varphi$, 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.

Table 2. MULICsoft results for clustering Mozilla with different similarity metrics.

Similarity metric	MoJo distance	Number of clusters	$\delta\varphi$
1st	424	156	80
2nd	417	227	80
3rd, x=6	399	187	110
3rd, x=7	397	183	110
3rd, x=9	388	191	130

As Table 2 shows, for all of our similarity metrics, the MoJo distance to the authoritative manual decomposition is significantly lower than the distances

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

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

of ACDC, BUNCH and LIMBO. The 3rd similarity metric produces the best results. For the 3rd similarity metric with x=9, the results are especially good with a MoJo distance of 388 to the authoritative decomposition. The reason why x=9 produces the best results is that the 3rd similarity metric amplifies significantly the effect of the high weights on the clustering process. We tried setting x to even higher values, such as 10, 12 and 15, 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 7 and 6, the results are still good, with MoJo distances of 397 and 399 respectively.

For the 1st and 2nd similarity metrics the results are better than those of ACDC, BUNCH and LIMBO. Note that for the 1st and 2nd similarity metrics a lower value of $\delta\varphi$ of 80 is used than for the 3rd metric. The reason for this is that with the 3rd metric 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 the 1st and 2nd metrics, on the other hand, fewer files are classified correctly during the first and second iterations and the lower value of $\delta\varphi$ allows some of the files to be considered instead at the next iterations.

5.2. MULICsoft with linear and exponential growths of φ

We also experimented with increasing the variable φ linearly by setting it to an initial value of 1 and increasing it by a constant value $\delta\varphi$ after each loop at which no object was placed in a cluster of size greater than one. We also experimented with increasing the variable φ exponentially by setting it 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.

Table 3 shows our results for both a linear and an exponential increase of φ. 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 assume no merging is done on the clusters after the clustering process.

Table 3. MULICsoft results for clustering Mozilla with linear and exponential growths of φ. The initial value of φ is 1.

MoJo dist.	Number of clusters	$\delta\varphi$	Similarity metric
Linear growth of φ, 3rd similarity metric, x=9			
399	187	150	3rd, x=9
391	188	140	3rd, x=9
388	191	130	3rd, x=9
408	197	110	3rd, x=9
410	199	90	3rd, x=9
Linear growth of φ, 3rd similarity metric, x=6			
402	187	120	3rd, x=6
399	187	110	3rd, x=6
399	197	100	3rd, x=6
407	195	105	3rd, x=6
402	199	90	3rd, x=6
412	207	80	3rd, x=6
414	212	70	3rd, x=6
Exponential growth of φ			
456	280	multiply φ by 2	3rd, x=9

As Table 3 shows, for the 3rd similarity metric a value of $\delta\varphi$ between 70 and 150 gives the best results overall, with MoJo distances as low as 388, 391 and 399. The reason why a high value of $\delta\varphi$ 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 the 3rd similarity metric and x=6 a value of $\delta\varphi$ between 100 and 110 gives the best results, while for x=9 a value of $\delta\varphi$ between 130 and 150 gives the best results. The reason for this is that with x=9 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=6, on the other hand, fewer files are classified correctly during the first and second iterations and the lower value of $\delta\varphi$ allows some of the files to be considered instead at the next iterations.

For the exponential growth of φ, the MoJo distance increases to 456. 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 the next section.

5.3. 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 4.2. 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 the 3rd similarity metric with x=7. We assume no merging is done on the clusters after the clustering process. Table 4 shows the results.

Table 4. MULICsoft results for setting a *threshold* for φ and treating some files as outliers. The initial value of φ is 1. The 3rd similarity metric is used with *x=7*.

$\delta\varphi$	Thres-hold for φ	MoJo dist.	Number of clusters	Number of outliers
Linear growth of φ				
120	121	651	110	402
120	121	477	180	141
10	50	627	254	287
10	80	523	276	126
50	51	604	214	266
60	61	582	207	243
99	100	508	179	152
Exponential growth of φ				
multiply φ by 2	32	672	250	352
multiply φ by 2	64	577	270	186

As Table 4 shows, the MoJo distance increases after treating objects as outliers. This increase in MoJo distance is related to the fact that each object that is treated as an outlier is placed in an independent cluster of size one. Thus, many Moves and Joins need to be performed for the computed decomposition to reach the authoritative manual decomposition and it is expected for the MoJo distance to increase. We do not interpret the increase in MoJo distance as a decline of the quality of the results, but as a sign that we should treat outliers in a different way from placing them in clusters independently. The MoJo distance would significantly decrease if all outliers were placed instead in one cluster together. Furthermore, the MoJo distance decreases even more if the outliers are simply ignored and the distance is computed between the intersection of files in the computed decomposition with files in the authoritative manual decomposition. A different distance measure could be useful for computing the distance between a computed decomposition and an authoritative decomposition, when outliers are involved.

5.4. Merging of clusters

MULICsoft provides the capability to merge clusters that are very similar after the clustering process, for the purpose of reducing the number of clusters, as described in Section 4.1. Table 5 presents the results after merging the clusters. For this experiment we ignore the files that are outliers, since they are classified independently in clusters of size one and are not merged.

Table 5. MULICsoft results after merging clusters. The initial value of φ is 1 and it grows exponentially by multiplying φ by 2. The threshold is 32. The 3rd similarity metric is used with *x=7*.

Clusters after merging	MoJo distance	Clusters	Files classified
100 clusters	Increased from 320 to 374	Reduced from 250 to 100	850
90 clusters	Increased from 320 to 379	Reduced from 250 to 90	850
80 clusters	Increased from 320 to 384	Reduced from 250 to 80	850

As Table 5 shows, in all cases the initial number of clusters is 250. We merge clusters until the number of clusters decreases to 100, 90 and 80. The MoJo distance increases, but this may be due to the MoJo distance metric not being able to capture some structural change in the clusters caused by the merging. In any case, the slight increase in MoJo distance caused by merging clusters is an interesting observation and we will be investigating it further. Perhaps the harm from the increase in MoJo distance is less than the benefit from decreasing the number of clusters and MoJo is not able to capture this tradeoff.

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. In case no developer was known, a unique dummy value for each file is used.
- **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 6 shows the MULICsoft MoJo distances to the authoritative decomposition for Mozilla, after inputting additional categorical data sets.

Table 6. MULICsoft results for clustering Mozilla with additional categorical data. The initial value of φ is 1 and it grows linearly. The 3rd similarity metric is used with *$x=9$. $\delta\varphi=130$.* Threshold has its default value.

Categorical Data Sets	MoJo distance	Number of clusters
Dev+Dir+LocEQ+Tim	387	196
Dev+Dir+LocEQ	407	192
Dev+Dir+Tim	407	192
Dev+Dir	409	177

As Table 6 shows, the MoJo distance to the authoritative decomposition does not significantly improve after inputting additional categorical data sets to MULICsoft. After inputting all 4 additional data sets of Dev+Dir+LocEQ+Tim, the result is 387, which is just slightly 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 increases slightly. For 3 additional categorical data sets the distance increases to 407 and for 2 additional categorical data sets the distance increases again to 409. The reason why we do not observe a significant improvement is likely to be that for the additional data sets we have set all weights to a default value of 1.0, implying that all of the additional data will influence the clustering process the same. However, in the additional data sets different data should have a different effect on the process. For example, a directory path with a large number of files should be more influential in clustering any of its files than a directory path with only a few files. These experiments are only preliminary, as we have not evaluated MULICsoft for all types and combinations of additional data sets and we need to assign different weights to the data.

7. Runtime Evaluation

Our experiments were performed on a Sun Ultra 60 with 256 MB of memory and a 300 MHz processor. Table 7 shows the run times it took for MULICsoft to cluster the files of the Mozilla system.

Table 7. MULICsoft runtimes, in seconds. The initial value of φ is 1 and it grows linearly.

Time (Seconds)	Sim. metric	$\delta\varphi$	Threshold
12	3rd, x=9	90	Default (1202)
12	3rd, x=9	120	Default (1202)
29	3rd, x=7	10	50
13	3rd, x=7	120	121
13	3rd, x=9	50	Default (1202)

8. Conclusion and Future Work

We have presented a software clustering algorithm, named MULICsoft. 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. It 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 have evaluated the quality of MULICsoft results on the Mozilla software system for which an expert-defined authoritative system decomposition exists. On this data set, the MULICsoft distance to the authoritative decomposition was lower than the distances of LIMBO [4], BUNCH [8] and ACDC [9]. Finally, we showed that the runtime of MULICsoft was satisfactory as it took between 10 and 30 seconds.

Future work will include improving the method for merging clusters that are similar, to build larger clusters, after the clustering process. We are currently implementing and testing an improved method for merging the clusters. This improved merging method will hopefully produce better MoJo distance results than the current one.

The results for inputting additional categorical data sets to MULICsoft are only preliminary. Thorough experiments with more types of data are needed. We intend to conduct more experiments for additional categorical data, such as data on the developers of the source files, the time stamps of source files, the directory paths of the source files, etc. Furthermore, we will be setting the weights for the additional data sets to values other than the default value of 1.0 as it is currently, depending on the effect that the data should have on the clustering process.

The results for identifying outliers indicate that often the distance to the authoritative decomposition increases when outliers are identified. However, this is related to the fact that in the computed decomposition each outlier is inserted in an independent cluster of size one. We will be experimenting with different ways to handle outliers, such as inserting them all in one cluster. This would decrease the MoJo distance, since fewer Moves and Joins would need to be performed in the computed decomposition to reach the authoritative manual decomposition. Furthermore, we are designing a different distance measure for computing the distance between a computed decomposition and an authoritative decomposition, when outliers are involved.

9. References

[1] B. Andreopoulos, A. An and X. Wang. (2005) BILCOM: Bi-level Clustering of Mixed Categorical and Numerical Biological Data. Technical Report # CS-2005-01. Department of Computer Science and Engineering, York University.

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

[3] P. Andritsos, V. Tzerpos. Information-Theoretic Software Clustering. IEEE Transactions on Software Engineering, Vol.31, No.2, February 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), March 2004.

[5] M.W. Godfrey and 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.

[6] Huang Z. (1998) Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical Values. Data Mining and Knowledge Discovery 2(3): 283-304.

[7] Brian S. Mitchell, Spiros Mancoridis. Comparing the Decompositions Produced by Software Clustering Algorithms using Similarity Measurements. In Proceedings of the International Conference on Software Maintenance (ICSM) 2001, pages 744-753.

[8] S. Mancoridis, B.S. Mitchell, Y. Chen, and 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, 1999.

[9] Vassilios Tzerpos and Richard C. Holt. ACDC: An algorithm for comprehension-driven clustering. In Proceedings of the Seventh Working Conference on Reverse Engineering, pages 258-267, 2000.

[10] Vassilios Tzerpos and Richard C. Holt. MoJo: A Distance Metric for Software Clusterings. In Proceedings of the Sixth Working Conference on Reverse Engineering, pages 187-, 1999.

[11] Vassilios Tzerpos and Richard C. Holt. The Orphan Adoption problem in Architecture Maintenance. In Proceedings of the Fourth Working Conference on Reverse Engineering 1997, Amsterdam, October 1997, pages 76-82.

[12] C. Xiao and V. Tzerpos. Software Clustering Based on Dynamic Dependencies. In Proceedings of the Ninth European Conference on Software Maintenance and Reengineering, to appear.