Multi-Layer Increasing Coherence Clustering of Large Software Data Sets with MULICsoft

Bill Andreopoulos

Aijun An

Xiaogang Wang

Technical Report CS-2005-06

April 2005

Department of Computer Science and Engineering

4700 Keele Street North York, Ontario M3J 1P3 Canada

Multi-Layer Increasing Coherence Clustering of Large Software Data Sets with MULICsoft

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

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

Abstract

We present the MULICsoft software clustering tool. This tool is intended for categorical data sets in which each categorical attribute value (CA) has a 'weight' in the range 0.0 to 1.0, indicating how strongly the corresponding CA should influence the clustering process. MULICsoft produces as many clusters as there naturally exist in the data set. Each cluster consists of layers formed gradually through iterations, by reducing the similarity criterion for inserting objects in layers of a cluster at different iterations. We have applied this tool to clustering the mozilla software system. The objects in the data are files. The CAs represent the relationships, such as invocation and dependency relationships, between files. The weights depend on the number of times that each file invoked other files during a run time profiling of execution. The results of MULICsoft are better than those of LIMBO, BUNCH and ACDC, as shown by the MoJo error rates that are derived by comparing the computed partitions of mozilla with an authoritative manual partitioning. http://www.cs.yorku.ca/~billa/MULIC/

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. Automatic clustering techniques generally employ certain criteria (i.e., low coupling and high cohesion) in order to decompose a software system into subsystems [Tzerpos00, Mancoridis99, Mancoridis01]. Manual decomposition of the system is done by software engineers. However, it takes plenty of time and it requires full knowledge of the system.

We developed the MULICsoft categorical clustering tool, which is based on the MULIC algorithm that is described in [Andreopoulos04]. 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 [Andreopoulos04]. Characteristics of MULIC and MULICsoft include: **a.** The algorithm does not sacrifice the coherence 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 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 discussed the runtime performance. Sections 8 and 9 discuss the results and conclude the paper.

2. Description of Data Sets

Both categorical and numerical data can be used for clustering a software system.

Categorical data. A categorical data set on a software system represents for each file, which other files it may invoke. Such data sets are referred to as *static* and exist in a market-basket format. Each row of the data set corresponds to a file *x* of the software system. In each row after the file name *x* there is a list of the other filenames that *x* may invoke during execution.

Numerical data. A numerical data set on a software system contains the results of a profiling of the execution of the system, representing how many times each file invoked other files during the run time. Such data sets are referred to as *dynamic*. Each row of the data set corresponds to a file *x* of the software system. In each row after the file name *x* there is a list of the other filenames that *x* invoked as well as how many times *x* invoked them, during the profiled run time.

We have applied the MULICsoft tool for clustering of large software systems to the mozilla system. There are 1202 objects in the mozilla data set, corresponding to 1202 source files of the mozilla system. Each object in the data set corresponds to a source file. We use both categorical and numerical data in clustering. The CAs are boolean values describing the relationships between the mozilla files. The CAs for an object represent the relationships, such as invocation and dependency relationships, of the corresponding source file with each of the other files. The data set contains a large number of 'zeros' with 'ones' occurring sparsely. We use numerical data in clustering by imposing a 'weight' in the range 0.0 to 1.0 on each CA, indicating how strongly the corresponding CA should influence the clustering process. We use a special similarity formula, as described in sections 4.4 and 5.1.

3. Related Work

Bunch is a clustering tool intended to aid the software developer and maintainer in understanding, verifying and maintaining a source code base [Mancoridis99]. The input to Bunch is a Module Dependency Graph (MDG). Figure 1 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 clusters (representing subsystems) and independent modules are assigned to separate clusters. Figure 1b shows a "good" partitioning of figure 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) [Mancoridis99].

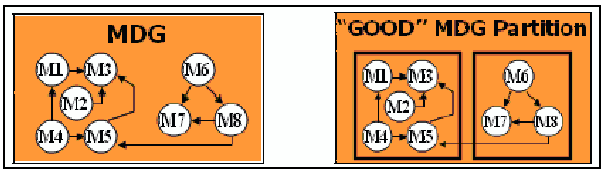

Figure 1. An MDG graph, from [Mancoridis99].

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 [Tzerpos00]. In the first one it creates a skeleton of the final decomposition by identifying subsystems using a pattern-driven approach. There are many patterns that have been used in ACDC [Tzerpos00]. 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. 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 [Andritsos04] as a scalable hierarchical categorical clustering algorithm that builds on the *Information Bottleneck (IB)* framework for quantifying the relevant information preserved when clustering. LIMBO uses the IB framework to define a distance measure for categorical tuples. LIMBO handles large data sets by producing a memory bounded summary model for the data.

4. The MULICsoft Clustering Algorithm

Clustering is performed using MULICsoft – an extension of MULIC [Andreopoulos04] – having a special similarity metric that is described in Sections 4.4 and 5.1. Figures 2 and 3 illustrate the algorithm and the results, respectively. MULICsoft is an extension of the k-Modes clustering algorithm for categorical data sets [Huang98]. k-Modes is a clustering algorithm that deals with categorical data [Huang98]. The k-Modes clustering algorithm requires the user to specify from the beginning 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 *i*th attribute in the cluster of objects.

MULICsoft makes substantial changes to the k-Modes algorithm. The purpose of our clustering algorithm is to maximize the following similarity formula *at each iteration individually*, while at the same time to ensure that all objects can eventually be inserted in clusters if so desired:

$$\sum_{i=1}^{N} similarity(o_i, \text{mode}_i) \qquad (1)$$

where o_i is the ith object in the data set and mode_i is the mode of the ith object's cluster containing the most frequent values in the cluster. Maximizing formula (1) leads to the situation where all objects are as similar to their clusters' modes as possible, thus minimizing the number of values in a cluster that differ from the most frequent value for the same attribute.

The MULICsoft algorithm has a few more requirements. First, the number of clusters is not specified by the user. Clusters should be formed, removed or merged during the clustering process as the need arises. Second, a cluster c must contain at least two objects, otherwise, it is not a cluster. Third, it must be possible for all objects to be inserted in clusters by the end of the clustering process if so desired.

MULICsoft normally is preceded by a preprocessing step, where the data objects are ordered according to the frequency by which their CAs appear in the data set – details of this step are given in [Andreopoulos04]. The main part of the MULICsoft clustering algorithm is shown in Figure 2. The algorithm starts by reading all objects from the input file and putting them in order. Then, it continues iterating over all objects that have not been placed in clusters yet, to find the closest cluster. In all iterations, the closest cluster for each unclassified object is determined by comparing how many similar attributes exist between a mode and the object. The similarity between a mode and an object is determined by the similarity equations described in Section 5.1.

The variable *num_values_can_differ* 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 *num_values_can_differ* equals 0, 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 are greater than *num_values_can_differ* then the object is inserted in a new cluster on its own. If the number of different values between the object and the closest cluster's mode are less than or equal to *num_values_can_differ*, then, the object is inserted in the closest cluster and the mode is updated.

At the end of each iteration, all objects classified in clusters with size one have their clusters removed so that they will be considered again at the next iteration. This ensures that the clusters that persist through the process are only those containing at least 2 objects for which a substantial similarity can be found. Objects belonging to clusters with size greater than one are removed from the set of unclassified objects.

At the end of each iteration, if no objects have been placed in clusters of size greater than 1, then the variable *num_values_can_differ* is incremented to represent how many values are allowed to differ next time. Thus, at the next iteration the threshold will be more flexible, ensuring that more objects will be placed in clusters. It is possible for all objects to be eventually classified, even if the closest cluster is a little similar. The iterative process stops when all objects have been placed in clusters of size greater than 1, or when *num_values_can_differ* exceeds a user-specified threshold.

The MULICsoft algorithm can eventually place all objects in clusters, because *num_values_can_differ* can continue increasing until all objects are classified. Even if, in the extreme case, an object o with I attributes has only one value similar to the mode of the closest cluster, it can still be classified when *num_values_can_differ* = *I-1*.

Input: (1) a set *S* of objects;
(2) the maximum number of values that can differ between an object and the mode of its cluster (*threshold* for *num_values_can_differ*)
Output: a set of clusters
Method:

1. Order the objects in *S* according to their scores [2];
2. Insert the highest-ordered object into a new cluster, use the object as the mode of the cluster, and remove the object from *S*;
3. Initialize *num_values_can_differ* to 0;
4. Loop through the following until S is empty or *num_values_can_differ* is greater than the specified *threshold*
   a. For each object *o* in *S* from the highest-ordered to lowest-ordered
      i. Find *o*'s closest cluster *c* by comparing 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 *num_values_can_differ*, 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 cluster with size > 1, increment *num_values_can_differ* by 1.

Figure 2. MULICsoft clustering algorithm

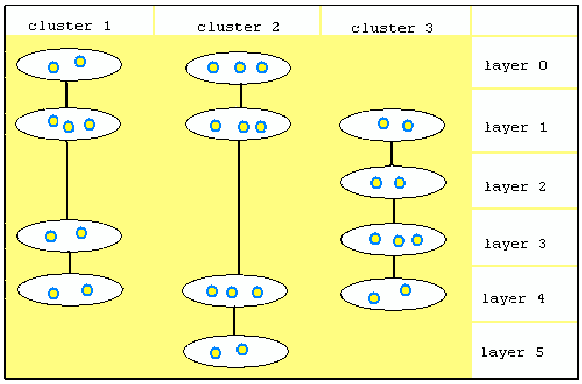

Figure 3. MULICsoft results. Each cluster consists of one or more different layers representing different coherences and similarities of the objects attached to the cluster.

Figure 3 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 allocated to it. The cluster's layer in which an object is inserted depends on the value of *num_values_can_differ*. Lower layers have a lower coherence and correspond to higher values of *num_values_can_differ* and to a more flexible similarity criterion for insertion. MULICsoft starts by inserting as many objects as possible in high layers – such as layer 0 or 1 - and then moves to lower layers, creating them as the need arises. Eventually, all objects can be classified in clusters and if little similarity exists between an object and its closest cluster mode, the object can be inserted in a low layer of 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 modes of the clusters' top layers. These modes were stored by MULICsoft when the clusters were created, so they do not need to be recomputed. If the object has equal similarity to all top layers' modes, 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 runtime complexity of MULICsoft is *O(IN)*, where *I* represents the number of annotations in an object and *N* represents the number of objects in the data set [Andreopoulos04].

4.1. Optional 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, at the end of 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. Besides increasing the cluster coherence, this process reduces the total number of clusters, as well as the resulting error rates, as described in Section 5.4. 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 ($Q_c = \{q_{c1}, \ldots, q_{cm}\}$ and $Q_d = \{q_{d1}, \ldots, q_{dm}\}$) is defined as:

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

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

4.2. Optimized Version of MULICsoft

We developed an optimized version of MULICsoft for runtime purposes. The optimized version increases the similarity criterion *num_values_can_differ* by 3 or 5 at a loop, while the non-optimized version increases it by 1 at a time. Sometimes - but not always - there is a slight loss in accuracy for the optimized version. On the other hand, sometimes there is a gain in accuracy, as Section 5.2 describes for clustering of software systems. The runtime is significantly better. Section 5.2 also describes increasing *num_values_can_differ* exponentially instead of linearly.

4.3. Dealing with Outliers

MULICsoft can eventually put all the objects in clusters. When *num_values_can_differ* equals the number of attributes, an unclassified object can be inserted in the lowest layer *h* of any existing 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 maximum value for *num_values_can_differ*, represented as *threshold* in Figure 2. When this value is reached, any remaining objects are not classified and are treated as outliers. As discussed in Section 5.3 for clustering of software systems, the overall quality of the results almost always improves by treating the lowest-layer objects as outliers.

4.4. MULICsoft Extensions to MULIC

MULICsoft includes extensions to MULIC for software clustering. A reason why software data needs a special clustering method is that the data contains mostly 'zeros' with 'ones' occurring sparsely.

A mode for a cluster contains 'zeros' by default. A position of the mode 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 zero at the corresponding position.

When calculating the similarity of a mode and an object, pairs of 'zero' attribute values between mode and object are ignored. The CAs in an object are represented as o_i.

All CAs in an object have "weights" in the range of 0.0 to 1.0 associated with them, which are dynamic data derived from profiling the execution of a system. We represent an object's weights as *f_object*. The weights were normalized in the data set, so that they span the range from 0.0 to 1.0, by dividing all numbers in a column by the maximum value in that column. Thus, there is at least one '1.0' value in each column of the data set.

Besides storing the boolean values of zero or one for the mode 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 refer to this special mode as the "real numbers' mode" and we represent its values as f_mode_i. These values are used by some of our similarity metrics to compute which cluster is the closest to an object by computing the similarity between the cluster's mode and the object.

Novel similarity metrics are defined to compute the similarity between a mode and an object, considering the CAs with their weights, as described in Section 5.1.

4.5. MoJo

The evaluation measure is called "MoJo distance". This distance is defined as the minimum number of Move and Join operations to transform one decomposition into another. Move and join operations are defined as:

Move: Remove an object from a cluster and put it in a different cluster.

Join: Merge two clusters into one.

Figure 4 shows two decompositions A and M. The MoJo distance between A and M is equal to 2. Since we have to *Move* object 4 to one of the bottom clusters and *Join* two bottom clusters in decomposition A in order to get to decomposition M [Tzerpos99, Mancoridis01].

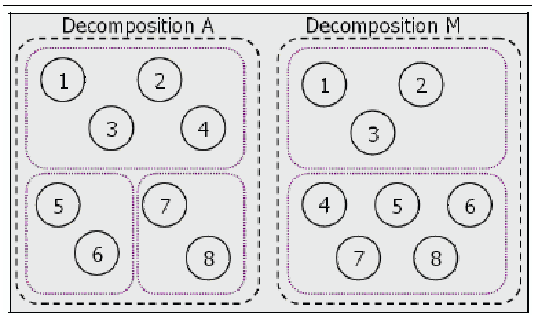

Figure 4. The MoJo distance between decompositions A and M.

5. Results for Clustering Mozilla with MULICsoft

MULICsoft clusters the 1202 mozilla files into 100-200 clusters before merging. The clusters produced for mozilla before merging had sizes ranging from 3 to 37 files. To evaluate the clustering result, we compared it with the authoritative decomposition, which is the human-made partition used as a reference for comparison. We measured the error rate using the MoJo distance metric, which is defined as the minimum number of "move" and "join" operations to transform a decomposition into the authoritative one. The results indicate that MULICsoft outperforms other software clustering tools, such as LIMBO [Andritsos04], BUNCH [Mancoridis99] and ACDC [Tzerpos00].

The MoJo error rates for ACDC, BUNCH and LIMBO applied to clustering the mozilla software system are shown in the table below. Other software clustering tools return MoJo error rates between 438-640 for clustering the mozilla system. MULICsoft partitioned the mozilla system into clusters of files, giving MoJo error rates such as 249, 320, 340, 388, 397, 399, 424 to 456 [Tzerpos99, Mancoridis01]. When clustering all files without treating any as outliers, the MULICsoft error is 388. MULICsoft can produce an error as low as 249, by setting the criterion *num_values_can_differ* to an initial value of 1, by increasing the criterion by 120 after each loop at which no file was placed in a cluster of size greater than one and by treating the 300 files in bottom layers as outliers.

	Mozilla		
	MoJo Error	**Number of Clusters**	**Objects classified**
ACDC	439	205	1202
BUNCH	440	21	1202
LIMBO	438	75	1202

5.1. Similarity metrics for comparison of objects to modes

A similarity metric is needed to choose the closest cluster to an object by computing the similarity between the cluster's mode and the object. We used the five similarity metrics described below, which consider the categorical attribute values and their weights. The weights were normalized in the data set - so that they were real values in the range from 0.0 to 1.0 – by dividing all numbers in a column by the maximum value in that column. Thus, there is at least one weight with a value of 1 in each column.

The first similarity metric uses the real numbers' mode values for each cluster, as described in Section 4.4, which are represented as *f_mode*. All similarity metrics use the weights of object *o*, which are represented as *f_object*.

Overall we found out that the 5th similarity formula is capable of producing the best results. This section describes the results for each of the 5 similarity formulas. We assume a linear increase of *num_values_can_differ* by setting it to an initial value of 1 and increasing it for the 1st and 2nd metrics by 80, for the 3rd and 4th metrics by 110, and for the 5th metric by 130, after each loop where no object was classified in a cluster of size greater than one. We set no upper bound – represented as *threshold* in Figure 2 - for *num_values_can_differ*, so that no objects are treated as outliers and all files are clustered. We do not merge the clusters at the end.

Similarity metric 1. The first similarity metric considers both the real numbers' mode values and the weights in each object, at pairs of categorical attribute values between an object *o* and a mode *m* that have identical values of 'one'.

$$f\_similarity(o,m) = \sum_{i=0}^{I} f\_objecti \times f\_\mathrm{mod}ei \times \delta(oi,mi)$$

$$\text{where } \delta(o_i, m_i) = \begin{cases} 1 & (o_i = m_i = 1); \\ 0 & otherwise \end{cases}$$

	Mozilla		
	MoJo Error	**Number of Clusters**	**Objects classified**
MULICsoft	424	156	1202

Similarity metric 2. The second similarity metric considers only the weights in each object, at pairs of categorical attribute values between an object *o* and a mode *m* that have identical values of 'one'.

$$f\_similarity(o,m) = \sum_{i=0}^{I} f\_objecti \times \delta(oi,mi)$$

$$\text{where } \delta(o_i, m_i) = \begin{cases} 1 & (o_i = m_i = 1); \\ 0 & otherwise \end{cases}$$

	Mozilla		
	MoJo Error	**Number of Clusters**	**Objects classified**
MULICsoft	417	227	1202

Similarity metric 3. The third similarity metric we used in this version of MULICsoft considers both the categorical attribute values and their weights. This metric amplifies the object positions having high weights, at pairs of CAs between an object *o* and a mode *m* that have identical values of 'one'.

$$f\_similarity(o,m) = \sum_{i=0}^{I} \frac{6-(4\times f\_objecti)}{5-(4\times f\_objecti)} \times \delta(oi,mi)$$

$$\text{where } \delta(o_i, m_i) = \begin{cases} 1 & (o_i = m_i = 1); \\ 0 & otherwise \end{cases}$$

This similarity metric places more importance on high weights (1.0) than low weights (0.1) at categorical attributes with identical values of 'one' between the object and the mode.

	Mozilla		
	MoJo Error	**Number of Clusters**	**Objects classified**
MULICsoft	399	187	1202

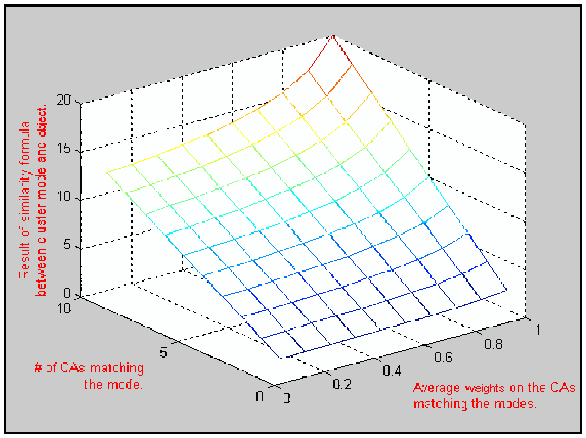

Figure 5. The function surface of *f_similarity*, using values for the weights between 0.0 and 1.0 as described previously.

Figure 5 shows the shape of the values returned by this similarity metric. Each object in this example has 10 CAs and weights. This graph shows that an object

will be much 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.

Similarity metric 4. The fourth similarity metric looks similar to the third similarity metric except that it amplifies even more the object positions having high weights, at pairs of categorical attribute values between an object *o* and a mode *m* that have identical values of 'one'.

$$f\_similarity(o,m) = \sum_{i=0}^{I} \frac{7-(4\times f\_objecti)}{5-(4\times f\_objecti)} \times \delta(oi,mi)$$

$$\text{where } \delta(o_i, m_i) = \begin{cases} 1 & (o_i = m_i = 1); \\ 0 & otherwise \end{cases}$$

The shape of the values returned by this similarity metric look similar to Figure 5, except that the *y-scale* values range from 0 to 30, instead of 0 to 20.

	Mozilla		
	MoJo Error	**Number of Clusters**	**Objects classified**
MULICsoft	397	183	1202

Similarity metric 5. The fifth similarity metric looks similar to the third and fourth similarity metrics except that it amplifies even more the object positions having high weights, at pairs of categorical attribute values between an object *o* and a mode *m* that have identical values of 'one'.

$$f\_similarity(o,m) = \sum_{i=0}^{I} \frac{9-(4\times f\_objecti)}{5-(4\times f\_objecti)} \times \delta(oi,mi)$$

$$\text{where } \delta(o_i, m_i) = \begin{cases} 1 & (o_i = m_i = 1); \\ 0 & otherwise \end{cases}$$

The shape of the values returned by this similarity metric look similar to Figure 5, except that the *y-scale* values range from 0 to 50, instead of 0 to 20.

	Mozilla		
	MoJo Error	**Number of Clusters**	**Objects classified**
MULICsoft	388	191	1202

5.2. Optimized MULICsoft: linear and exponential growths of *num_values_can_differ*

We experimented with increasing the variable *num_values_can_differ* linearly (0,3,6,9,12) by setting it to an initial value of x and increasing it by a constant value after each loop at which no object was placed in a cluster of size greater than one. We also experimented with increasing the variable *num_values_can_differ* exponentially (1,2,4,8,16) 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, as discussed in Section 4.2.

Specifically, our results are shown below for both a linear and an exponential increase of *num_values_can_differ*. We cluster all objects, without treating any of the objects in bottom layers as outliers. We use the 1st or 2nd or 3rd or 4th or 5th similarity metrics, as indicated below. We assume no merging is done on the clusters after the clustering process.

	Mozilla		
	MoJo Error	**Number of Clusters**	**Objects classified**
MULICsoft Linear growth			
- Initial value 1, increment by 150, 5th similarity metric	399	187	1202
- Initial value 1, increment by 140, 5th similarity metric	391	188	1202
- Initial value 1, increment by 130, 5th similarity metric	388	191	1202
- Initial value 1, increment by 110, 5th similarity metric	408	197	1202
- Initial value 1, increment by 90, 5th similarity metric	410	199	1202
- Initial value 1, increment by 110, 4th similarity metric	397	183	1202
- Initial value 1, increment by 100, 4th similarity metric	413	178	1202
- Initial value 1, increment by 100, 3rd similarity metric	399	197	1202

- Initial value 1, increment by 105, 3rd similarity metric	407	195	1202
- Initial value 1, increment by 110, 3rd similarity metric	399	187	1202
- Initial value 1, increment by 70, 3rd similarity metric	414	212	1202
- Initial value 1, increment by 80, 3rd similarity metric	412	207	1202
- Initial value 1, increment by 90, 3rd similarity metric	402	199	1202
- Initial value 1, increment by 120, 3rd similarity metric	402	187	1202
- Initial value 1, increment by 80, 2nd similarity metric	417	227	1202
- Initial value 1, increment by 75, 1st similarity metric	427	158	1202
- Initial value 1, increment by 80, 1st similarity metric	424	156	1202
MULICsoft Exponential growth			
- Initial value 1, multiply by 2, 5th similarity metric	456	280	1202

5.3. Treating objects as outliers by fixing an upper bound for *num_values_can_differ*

We have found that by treating the last 100-300 objects in bottom layers of clusters as outliers (from a data set of 1202) the error rate decreases significantly. Objects are treated as outliers by setting an upper bound – represented as *threshold* in Figure 2 - for the variable *num_values_can_differ*, as discussed in Section 4.3. For example, setting a *threshold* for *num_values_can_differ* of 150 means that clustering will stop at layer 150 and any objects in layers greater than 150 will be treated as outliers. This supports the idea that lower layers are less reliable than higher layers. We have experimented with various *thresholds* for *num_values_can_differ*, for both linear and exponential growths of *num_values_can_differ* as discussed in Section 4.2. We have used the 4th similarity metric as described in Section 5.1. We assume no merging is done on the clusters after the clustering process.

	Mozilla		
	MoJo Error	**Clusters**	**Objects classified**
MULICsoft Exponential growth, 4th similarity metric			
- Initial value 1, multiply by 2, threshold 32	320	250	850
- Initial value 1, multiply by 2, threshold 64	391	270	1016
MULICsoft Linear growth, 4th similarity metric			
- Initial value 1, increment by 120, threshold 121 (400 objects in bottom layers are treated as outliers)	249	110	800
- Initial value 1, increment by 120, threshold 121	336	180	1061
- Initial value 1, increment by 10, threshold 50	340	254	915
- Initial value 1, increment by 10, threshold 80	397	276	1076
- Initial value 1, increment by 50, threshold 51	338	214	936
- Initial value 1, increment by 60, threshold 61	339	207	959
- Initial value 1, increment by 99, threshold 100	356	179	1050

The results improve when increasing *num_values_can_differ* exponentially. We set its initial value to 1 and multiply its value by 2 after each loop at

which no file was placed in a cluster of size greater than 1. With a maximum *threshold* of 32 the error rate is 320 and 850 files are clustered. With a *threshold* of 64 the error rate is 391.

5.4. Optional merging of clusters

This algorithm 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. This section describes the results for merging the clusters, while modifying the parameters described above. We use the 4th similarity metric.

	Mozilla		
MULICsoft Exponential growth, 4th similarity metric	**MoJo Error**	**Clusters**	**Objects classified**
- 100 clusters after merging, Initial value 1, multiply by 2, threshold 32	Increased from 320 to 374	Reduced from 250 to 100	850
- 90 clusters after merging, Initial value 1, multiply by 2, threshold 32	Increased from 320 to 379	Reduced from 250 to 90	850
- 80 clusters after merging, Initial value 1, multiply by 2, threshold 32	Increased from 320 to 384	Reduced from 250 to 80	850

6. Inputting Additional Categorical Data

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

Dev	The names of all the developers who worked on each mozilla file.
Dir	The directories that contain each mozilla file, including the ancestors in the directory hierarchy.
LocEQ	The RANGE of each mozilla file.
Tim	The date at which each mozilla file was developed.

The MULICsoft error rates for mozilla are described below, after inputting additional categorical data sets.

	Mozilla with Dev+Dir+LocEQ+Tim	
MULICsoft linear growth, 5th similarity metric	**MoJo Error**	**Number of Clusters**
- Initial value 1, increment by 130, NO threshold	387	196

	Mozilla with Dev+Dir+LocEQ	
MULICsoft linear growth, 5th similarity metric	**MoJo Error**	**Number of Clusters**
- Initial value 1, increment by 130, NO threshold	407	192

	Mozilla with Dev+Dir+Tim	
MULICsoft linear growth, 5th similarity metric	**MoJo Error**	**Number of Clusters**
- Initial value 1, increment by 130, NO threshold	407	192

	Mozilla with Dev+Dir	
MULICsoft linear growth, 5th similarity metric	**MoJo Error**	**Number of Clusters**
- Initial value 1, increment by 130, NO threshold	409	177

7. Runtime Evaluation

The run times taken for MULICsoft to cluster the mozilla system are shown in the table below.

	Time (Seconds)
MULICsoft linear growth, 5th similarity metric - Initial value 1, increment by 90, NO threshold	12
MULICsoft linear growth, 5th similarity metric - Initial value 1, increment by 120, NO threshold	12
MULICsoft linear growth, 4th similarity metric - Initial value 1, increment by 10, threshold 50	29

MULICsoft linear growth, 4th similarity metric - Initial value 1, increment by 120, threshold 121	13
MULICsoft linear growth, 5th similarity metric - Initial value 1, increment by 50, NO threshold	13

8. Discussion

MULICsoft offers many advantages for clustering of software data sets. MULICsoft allows for a flexible number of clusters to be produced. A MULICsoft cluster is much more representative of the underlying patterns in a data set, because MULICsoft recognizes that differing layers of coherence and similarity may exist in a cluster amongst objects. We have shown that when requiring the user to specify the number of clusters to be output from the clustering process, this might not allow the clusters to have the maximum coherence [Andreopoulos04].

Our method provides the user with the option to merge any clusters that are very 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 further improve the results of MULICsoft, because sometimes the results might be refined.

Human comprehension of a software system may benefit from a clustering method that initially focuses on creating clusters of highly coherent objects. A software system is typically composed of many tight and highly coherent clusters of files and a clustering method should start by identifying as many of these highly coherent clusters as possible. If a manual decomposition of the software system identifies 10 clusters, it does not mean that this decomposition is the best nor the most useful when trying to comprehend or maintain the program, since the resulting clusters may have a large size. On the other hand, a clustering partition resulting in smaller clusters of highly coherent files may be much more beneficial to a software maintainer.

9. Conclusion

We have presented a clustering algorithm, named *MULICsoft*, that deals with many of the problems posed by k-Modes. Specifically, MULICsoft does not sacrifice the coherence of the data sets for the number of clusters, which in k-Modes is defined strictly before the process. Instead, MULICsoft finds the clusters that naturally exist in the data set and forms as many clusters as required. For each cluster, MULICsoft can form layers of varying coherence. It starts by forming a layer of high coherence using strict criteria concerning which objects to insert in the layer. As the process continues MULICsoft relaxes its criteria, forming layers that may be less coherent than the previous layers. In addition, MULICsoft similarity metrics consider weights on the categorical attribute values 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 tested MULICsoft for accuracy on the mozilla system for which an expert-defined system partition exists. On this data set the MULICsoft misclassification rate was lower than the misclassification rates of LIMBO [Andritsos04], BUNCH [Mancoridis99] and ACDC [Tzerpos00]. Finally, we showed that the runtime of MULICsoft which took between 10 and 30 seconds was more than satisfactory.

10. Acknowledgements

The authors thank Professor Vassilios Tzerpos for providing the mozilla data set and the MoJo clustering error rates for ACDC, BUNCH and LIMBO.

11. 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, York University.

[2] B. Andreopoulos, A. An and X. Wang. (2004) MULIC: Multi-Layer Increasing Coherence Clustering of Categorical Data Sets. Department of Computer Science, York University. Submitted to KDD 2005.

[3] P. Andritsos, P. Tsaparas, R. J. Miller, K. C. Sevcik. LIMBO: Scalable Clustering of Categorical Data. In 9th International Conference on Extending DataBase Technology (EDBT), March 2004.

[4] 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.

[5] Brian S. Mitchell, Spiros Mancoridis, Comparing the Decompositions Produced by Software Clustering Algorithms using Similarity Measurements (2001).

[6] 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 Proc. Of International Conference on Software Engineering, 1999.

[7] 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.

[8] 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.