# MULIC:Multi-Layer Increasing Coherence Clustering of Categorical Data Sets

**Bill Andreopoulos**

**Aijun An**

**Xiaogang Wang**

Technical Report CS-2004-07

December 2004

Department of Computer Science and Engineering

4700 Keele Street North York, Ontario M3J 1P3 Canada

# MULIC: MULTI-LAYER INCREASING COHERENCE CLUSTERING OF CATEGORICAL DATA SETS

Bill Andreopoulos
Department of Computer Science, York University, Toronto, Ontario, Canada, M3J 1P3
billa@cs.yorku.ca

Aijun An
Department of Computer Science, York University, Toronto, Ontario, Canada, M3J 1P3
aan@cs.yorku.ca

Xiaogang Wang
Department of Mathematics and Statistics, York University, Toronto, Ontario, Canada,M3J1P3
stevenw@mathstat.yorku.ca

December 2004

## ABSTRACT

We present the *MULIC* algorithm for clustering of categorical data sets that offers major improvements over many aspects of the traditional k-Modes algorithm, so that the results are more accurate. A preprocessing of the objects in the data set is performed, that imposes an ordering of the objects. MULIC does not sacrifice the coherence of the resulting clusters for the number of clusters desired. Instead, it produces as many clusters as there seem to 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 show that the misclassification rates – including HA Indexes [16]- of MULIC are much lower than those of other algorithms, including k-Modes [14], ROCK [10], AutoClass [19] and the WEKA clustering algorithms [18,20]. We compare the MULIC run times to those of other algorithms, showing that MULIC has comparable or better run times. http://www.cs.yorku.ca/~billa/MULIC/

**Keywords:** Clustering, Categorical, Multi-layer, Coherence, HA Index, Data Mining

## 1. INTRODUCTION

Clustering attempts to partition a set of objects into groups, so that objects with similar characteristics are grouped together and different groups contain objects with different characteristics [5, 8,9,11]. K-Modes is a widely used categorical clustering algorithm that assigns a mode to each cluster as a 'summary' of the cluster's contents. Using a mode for each cluster during the clustering process ensures that a cluster will contain many categorical annotations of the same type. We were motivated to design a novel algorithm called *MULIC: Multi-Layer Increasing Coherence clustering of categorical data sets*, to overcome the following shortcomings of the classical k-Modes algorithm: **a.** the accuracy of k-Modes is often questionable. **b.** the results produced are usually subject to a strictly defined number of clusters. **c.** when the ordering of the input objects to the k-Modes clustering algorithm changes, the k-Modes produces different results. It is not the case that just because two different orderings produce different results with k-Modes, one ordering is more correct than the other, nor that one result is more correct than the other. If two different random orderings produce different results then we cannot claim that one result is better than the other.

Various ideas for improving k-Modes arose from our imposing a standard ordering on the objects. While ordering the objects in a reasonable way, questions arose in our mind, such as "why do modes represent the annotations that occur most frequently", "why aren't objects with categorical annotations (CAs) that occur more frequently than the CAs of other objects clustered differently", "why sacrifice the coherence of the initial clusters for the number of clusters desired", "why not create new clusters as the need arises during the process, instead of creating them all at the beginning".

*MULIC* has the following characteristics:

1) MULIC always produces the same results. We have placed an ordering on the objects and we claim that this is better than a random ordering. MULIC starts by clustering objects with high values because that way more objects are likely to be clustered early in the process.
2) MULIC does not require the user to specify the number of clusters before the process of clustering the data set – the number of clusters produced can change during the process. MULIC does not sacrifice the coherence of the initial clusters for the number of clusters desired. MULIC prefers to form as many clusters of high coherence as can be found naturally in the data set.
3) MULIC forms clusters gradually and if a new pattern is discovered, then, MULIC allows for a new cluster to be formed.
4) MULIC clusters consist of many layers, where each layer contains objects of different similarity and coherence. Higher layers in a cluster are created earlier in the process and are more coherent – i.e. the objects are more similar to one another - than lower layers that are created later.
5) MULIC ensures that the initial layers created will be as coherent as possible, meaning that early in the process 2 objects must have at most 1 different annotation for them to be inserted in the same cluster. The layers that are formed at the end of the process contain the objects that MULIC had difficulty clustering earlier because they did not have enough similarity.
6) MULIC offers the user the opportunity to modify the resulting clusters in the end, by merging similar clusters to reduce their number.

This paper is organized as follows. First, in Section 2, we describe previous work on clustering and k-Modes. In Sections 3 and 4 we describe MULIC and we explain how the design meets the goals of clustering and analysis. In Section 5 we describe the results for testing the accuracy and runtime performance of MULIC against other algorithms, including k-Modes, ROCK and AutoClass. A computational complexity analysis is also discussed in Section 5. We discuss our main contributions in Section 6 and we conclude the paper in Section 7.

## 2. RELATED WORK

A few algorithms have been proposed in recent years for clustering categorical data. Some of these algorithms, such as AutoClass and ROCK, do not require the user to specify the number of clusters *k* beforehand.

AutoClass is a clustering algorithm that can work on categorical as well as numerical data sets [19]. AutoClass uses a Bayesian method for determining the optimal classes. AutoClass takes a prior distribution of each attribute in each cluster, symbolizing the prior beliefs of the user. It changes the classifications of items in clusters and changes the mean and variance of the distributions, until the mean and variance stabilize.

ROCK, an adaptation of an agglomerative hierarchical clustering algorithm, is introduced in [10]. ROCK assumes a similarity measure between tuples and defines a link between two tuples whose similarity exceeds a threshold *w*. Initially, each tuple is assigned to a separate cluster and then clusters are merged repeatedly according to the closeness between clusters. The closeness between clusters is defined as the sum of the number of "links" between all pairs of tuples, where the number of "links" represents the number of common neighbors between two clusters.

k-Modes was introduced in [14] and described in Section 2.1. An extension of k-Modes called k-Prototypes was proposed in [13] for dealing with mixed numerical and categorical data sets. K-Prototypes also uses an iterative approach to clustering that continues until objects stop changing clusters.

A fuzzy k-Modes algorithm was proposed in [15]. In real applications there is very often no sharp boundary between clusters so that fuzzy clustering is often better suited for the data. Membership degrees between zero and one are used in fuzzy clustering instead of crisp assigments of the data to clusters.

LIMBO is introduced in [3], 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.

COOLCAT is introduced in [4] as an entropy-based algorithm for categorical clustering. Clusters are created by "cooling" them down, i.e. reducing their entropy. COOLCAT relies on sampling and is non-hierarchical. COOLCAT starts with a sample of points and identifies a set of *k* initial tuples such that the minimum pairwise distance among them is maximized. All remaining tuples of the data set are placed in one of the clusters such that, at each step, the increase in the entropy of the resulting clustering is minimized.

STIRR is introduced in [7] as an iterative algorithm based on non-linear dynamical systems. STIRR applies a linear dynamical system over multiple copies of a hypergraph of weighted attribute values, until a fixed point is reached. Each copy of the hypergraph contains two groups of attribute values - one with positive and another with negative weights - which define the two clusters. The approach used in STIRR can be mapped to certain types of non-linear systems.

Squeezer is introduced in [12] as a one-pass algorithm. Squeezer repeatedly reads tuples from the data set one by one. When the first tuple arrives, it forms a cluster alone. The consequent tuples are either put into an existing cluster or rejected by all existing clusters to form a new cluster by the given similarity function.

CLOPE is introduced in [21] as a clustering algorithm for categorical and transactional data. CLOPE uses a heuristic method of increasing the height-to-width ratio of the cluster histogram.

CACTUS is presented in [6], introducing a novel formalization of a cluster for categorical attributes by generalizing a definition of a cluster for numerical data. CACTUS is a summation-based algorithm that discovers exactly such clusters in the data. CACTUS consists of three phases: *summarization*, *clustering* and *validation*.

## 2.1 Background on k-Modes

K-Modes is a clustering algorithm that deals with categorical annotations (CAs) only [14]. The k-Modes clustering algorithm for categorical data sets 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.

During the k-Modes clustering algorithm for categorical data sets, the following generic loop is performed and most of its steps are redefined in MULIC:

```
Insert the first K objects into K new clusters.
Calculate the initial K modes for K clusters.
Repeat {
      For (each object O) {
      Calculate the similarity between object O
and the modes of all clusters.
      Insert object O into the cluster C whose
mode is the most similar to object O.
      }
      Recalculate the cluster modes so that the
cluster similarity between mode and objects is
maximized.
} until (no or few objects change clusters).
```

Each cluster has a mode associated with it. Modes are used to choose the closest cluster to an object by computing the similarity between the cluster's mode and the object. In the loop above, the object is then allocated to the closest cluster and the mode gets updated.

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. Assume that each object is described by *m* attributes. Let $X=\{x_1,x_2,\ldots,x_m\}$ be an object, where $x_i$ is the value for the ith attribute, and $Q=\{q_1,q_2,\ldots,q_m\}$ be the mode of a cluster. The similarity between X and Q is defined as:

$$\text{similarity}(X,Q)=\sum_{j=1}^{m}\delta(x_j,q_j)$$

*where* $\delta(x_j,q_j)=1$ *if* $x_j=q_j$, *0 otherwise.*

A mode $Q$ for a cluster $C$ is found by maximizing $\sum_{i=1}^{n} similarity(X_i,Q)$, which is maximized if and only if $\text{frequency}(X_j=q_j|C) >= \text{frequency}(X_j=c_j|C)$ for $q_j \neq c_j$ for all j=1 to m. $\text{Frequency}(X_j=q_j|C)$ is the number of objects in the cluster C that have the value $q_j$ in the jth attribute $X_j$. Thus, $\text{frequency}(X_j=q_j|C)$ must be maximal for all j=1 to m.

In the descriptions that follow we assume that *C* represents the total number of clusters and we use *c* to index the clusters. We assume *N* represents the number of objects and we use *n* to index the objects. $\text{Mode}_n$ represents the mode of the cluster in which object *n* is classified. We assume *K* represents the total number of iterations and we use *k* to index the iterations. We assume *I* represents the number of annotations in an object and we use *i* to index the annotations.

## 3. MOTIVATION FOR PROPROCESSING OF DATA SETS BEFORE CLUSTERING

We defined the "Pessimistic Error" metric for clustering of categorical data sets. This metric is based on the idea that most clustering attempts will result in clusters whose objects' annotations are not perfectly uniform in one or more annotation indexes. By this point of view, the clustering result will almost never be perfect. We proposed a formula for an error metric that a categorical clustering algorithm should minimize:

**Pessimistic Error metric:** Let $A_{ci1}$ be the *most frequently* occurring annotation at index *i* of the objects in cluster c, if such an annotation exists. Let $A_{ci2}..A_{ciM}$ be the rest of the annotations in cluster *c* at annotation index *i*, if such annotations exist. A clustering result for a categorical data set should minimize the pessimistic error metric:

$$pessimistic\_error\_metric = \sum_{c=1}^{C}\sum_{i=1}^{I}(\sum_{a=Aci2}^{AciM}1) \qquad (1)$$

where c∈ {1..C} represents all clusters, i∈ {1..I} represents all annotation indexes of the objects in a given cluster and a∈ $\{A_{ci2}..A_{ciM}\}$ represents the least frequently occurring annotations at index *i* of the objects in cluster *c*.

A categorical clustering algorithm should minimize formula (1). Usually, when some clustering algorithm succeeds in producing the most desired clustering result for the objects of a data set, one or more of the clusters do not have uniform annotations in one or more of the annotation indexes of the objects classified in that cluster, so formula (1) does not return zero. Formula (1) only returns zero if the desired clusters have uniform categorical annotations across all of the member objects' annotation indexes. However, such a case would be trivial.

By preprocessing the data sets as we describe in Section 4.1 we attempt to minimize the return value of formula (1). This happens by ensuring that the objects presented to the clustering algorithm first - the objects contributing most to the "shaping" of the clusters - contain the most frequently occurring annotations in the data set. Thus, clusters are "shaped" on the basis of the most frequently occurring annotations rather than the seldom occurring annotations in the data set.

This is especially useful for a data set looking like the one in Figure 1 below, where we want class 2 objects to be clustered in different clusters from class 1 objects. As shown, class 1 contains fewer objects than class 2. The gray areas in classes 1 and 2 show that there are many objects in class 1 the categorical annotations of which overlap significantly with the annotations of some objects in class 2. The white areas in classes 1 and 2 show that the objects in those areas have distinct annotations between the two classes. If objects in class 1 were clustered before class 2 with k-Modes, then the mode of the first resulting cluster(s) would largely represent the annotations of class 1 in the gray area. Then many objects in the gray area of class 2 would end up erroneously being clustered in the same cluster as class 1 objects. On the other hand, if the objects were ordered by decreasing frequency of their annotations occurring in the data set, then objects in the white area of class 2 would be clustered first, then objects in the gray area of class 2 and then objects in class 1. In this case, the mode of the first resulting cluster(s) would not represent the annotations of the objects in the gray area of class 2, since objects in the white area of class 2 with more frequent annotations were clustered first. So it is unlikely that class 1 objects would be erroneously clustered with class 2 objects, since the mode of the first resulting cluster(s) represents primarily class 2 white area annotations.

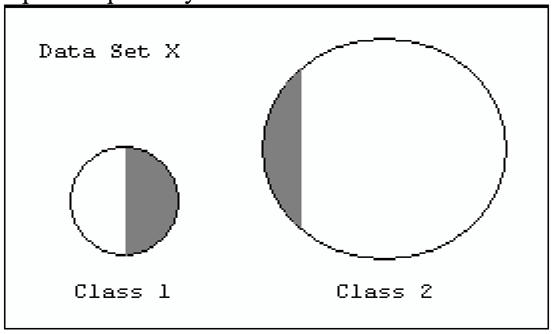


**Fig. 1 – A data set X consists of classes 1 and 2. The gray areas in classes 1 and 2 show that some of the categorical annotations of the objects in those areas overlap between the two classes. The objects in the white areas do not have overlapping annotations.**

## 4. THE *MULIC* CATEGORICAL CLUSTERING ALGORITHM

We developed a variation of the k-Modes clustering algorithm, called *MULIC*, that has many advantages over the traditional k-Modes. As with k-Modes, a prominent element in the clustering process is the "mode" for each cluster. The mode is a vector of categorical attributes whose size is equal to the number of categorical attributes in each object. Each index position in a mode represents which attribute occurs most frequently at that position over all objects in the cluster.

### 4.1 Data Pre-processing

"Modes" inspired us to arrange the data objects in an order before clustering them. Each different categorical annotation in the data set is 'counted' to determine how many times it occurs across all objects in the data set. For each object we sum the counts for all of the object's annotations and we assign the resulting 'value' to the object. The value indicates how frequently an object's annotations appear in the data set. Then, we order the objects by decreasing value and present the objects to MULIC in that order. This ensures that the modes in each cluster will be influenced more by the attributes that occur more frequently over all objects. Thus, if early in the process an object is compared to a mode and some of the object's attributes match the mode's attributes, then those attributes are likely to occur frequently in the objects that will be clustered in the next iterations.

Suppose that after setting the ordering on the objects, we start by inputting to the clustering process the object *X = {x1,x2,x3,x4}* with categorical annotations *x1="A", x2="V", x3="E"* and *x4="Q"*. The object *X* has a value of *i*. Then,

*value(X) = i = count(x1) + count(x2) + count(x3) + count(x4) = count("A") + count("V") + count("E") + count("Q").*

The counts above are the frequencies by which the annotations occur over all objects in the set. For the next object in the ordering, *Y={y1,y2,y3,y4}*:

*value(Y) <= i* .

if *value(Y) = i* then there exist only few combinations of annotations whose sum of counts equals *i*, so it will be likely for *y1=x1="A"*, *y2=x2="V"*, *y3=x3="E"* and *y4=x4="Q"*. Then, *Y* will be inserted in the same cluster as *X*.

if *value(Y) < i* then it is likely that 3 of the indexes of *Y* have categorical annotations that are the same as *X* and the 4th index of *Y* has an annotation different from *X* whose count is slightly lower; as a result *value(Y) < i*. Then, *Y* will be likely to be inserted in the same cluster as *X*.

If many positions of *Y* have annotations that are different from *X*, then *Y* might be inserted in a different cluster from *X*. However, it is likely that the object *K* that comes after Y in the ordering will have 3 or more annotations that are the same as *X* – then, *K* will be inserted in the same cluster as *X*. It is likely that *Y* will be inserted in the same cluster as *X*, but even if *Y* is inserted in a different cluster from *X*, it is likely that the object *K* after *Y* will be inserted in the same cluster as *X*.

This way, we avoid the situation where a cluster's mode is shaped early in the clustering process by annotations that occur infrequently in the data set, but just happened to be occurring frequently in the initial objects presented to the clustering process. Such annotations occurring frequently in a small subset of the objects might be what forms 'outliers' and thus we would not want them to overly influence the early stages of the clustering process. If this was the case, then later on in the process objects might be clustered incorrectly.

As an illustration of the undesirable situation that our algorithm helps to avoid, suppose that in the large cluster *ClusA* that we would like our clustering algorithm to produce, all member objects have the following annotations in the first 3 indexes:
*{'B', 'V', 'E', x4}*
In the 4th index *x4* there may exist various annotations, including *'I'*. If the objects are not ordered by the frequency of annotations, then the cluster *ClusB* might be formed early in the clustering process, consisting of 5 objects with the following annotations:
*{'S', 'V', 'E', 'O'}, {'S', 'V', 'E', 'I'}, {'X', 'V', 'E', 'I'}, {'S', 'B', 'E', 'I'}, {'S', 'B', 'A', 'I'}*
Then, the resulting mode for *ClusB* will have values *{'S', 'V', 'E', 'I'}*. If the objects are not ordered, then objects with annotations of *{'B', 'V', 'E', 'I'}* that really belong in *ClusA* will be classified in *ClusB* instead. But if the objects are ordered then *ClusA* will be formed first and *ClusB* last.

## 4.2 Clustering Algorithm

This use of count values for objects and categorical annotations also prompted us to make other 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 ensuring that all objects can eventually be inserted in clusters:

$$\sum_{n=1}^{N} similarity(object_n, mode_n) \qquad (2)$$

Object$_n$ is the nth object in the data set and mode$_n$ is the mode of the object's cluster containing the most frequent annotations in the cluster. Maximizing formula (2) minimizes the pessimistic error metric shown in formula (1) because all objects are as similar to their clusters' modes as possible, thus minimizing the number of annotations in a cluster that differ from the most frequent annotation for the same index position.

The MULIC algorithm has a few more important requirements: **a.** A cluster *C* must contain at least 2 objects to qualify as a cluster. **b.** It must be possible for all objects to be inserted in clusters by the end of the clustering process.

The MULIC clustering algorithm is shown in Figure 2. The algorithm starts by reading all objects from the input file and putting them in order, as described in Section 4.1. 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 is determined for each unclassified object 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 equation described in Section 2.1.

The variable *num_CAs_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_CAs_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 annotations between the object and the closest cluster's mode are greater than *num_CAs_can_differ* then the object is inserted in a new cluster on its own. If the number of different annotations between the object and the closest cluster's mode are less than or equal to *num_CAs_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 linked list of objects.

At the end of each iteration if no objects have been placed in clusters of size greater than 1, then the threshold *num_CAs_can_differ* is incremented to represent how many annotations 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_CAs_can_differ* is greater than a user-specified threshold.

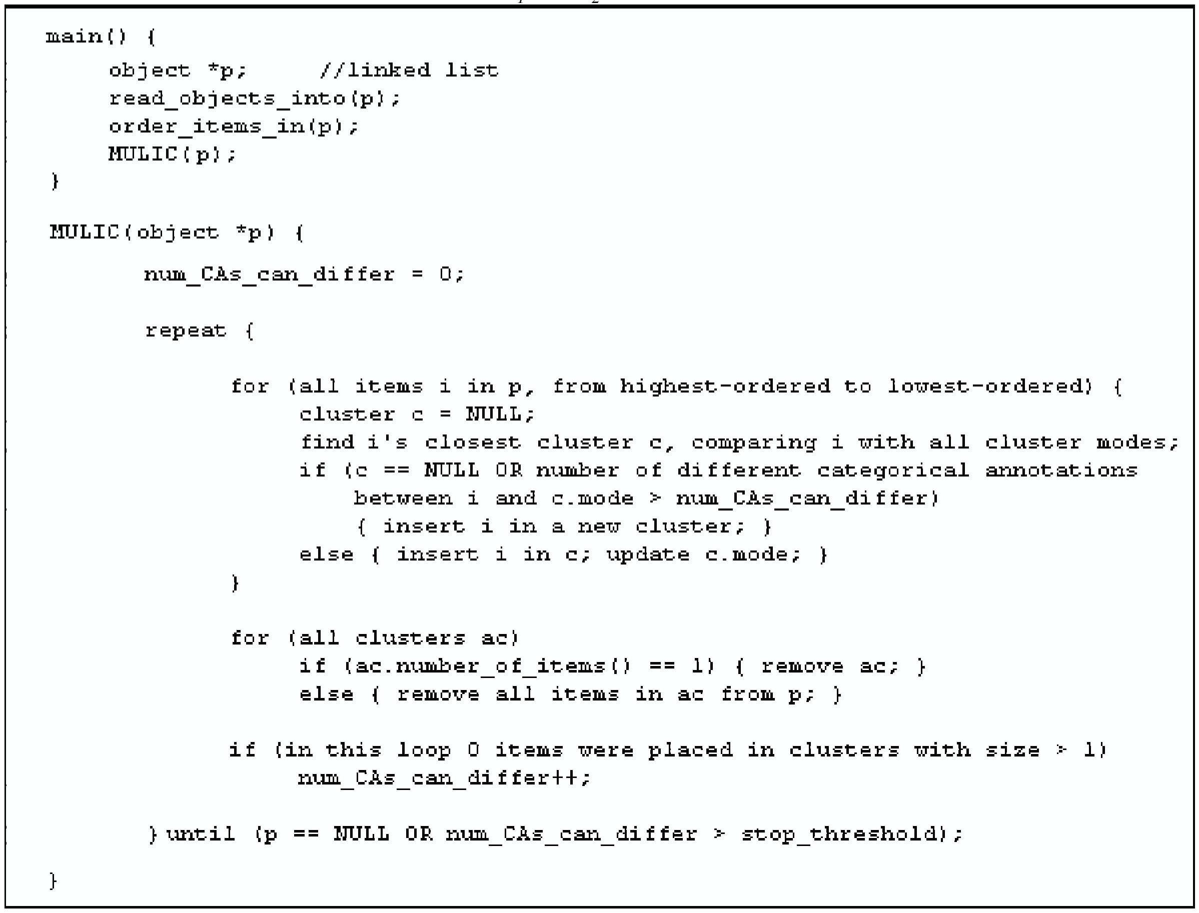
The MULIC algorithm of Figure 2 can eventually place all objects in clusters, because num_CAs_can_differ can continue increasing until all objects are classified. Even if, in the extreme case, an object *o* with *I* annotations has only one annotation similar to the mode of the closest cluster, it can still be classified when *num_CAs_can_differ = I-1.*

Figure 3 illustrates what the results of MULIC will typically 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_CAs_can_differ*. Lower layers have a lower coherence and correspond to higher values of *num_CAs_can_differ* and to a more flexible similarity criterion for insertion. MULIC starts by inserting as many objects as possible in high layers – such as layers 0 and 1 - and then moves to lower layers, creating them as the need arises. Eventually, all objects can be classified in clusters; 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 MULIC 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.

One might ask how we know that *(2)* will be maximized. Our ordering of the objects combined with the MULIC clustering process contributes to preventing various cases described in Sections 3 and 4.1. For instance, when objects from a smaller and a larger class have overlapping annotations the objects from the larger class will be presented first to the clustering process, so the first cluster(s) formed will contain mostly objects from the larger class. Furthermore, suppose a class consists of objects $o_1$, $o_2$, $o_3$ … $o_n$. All objects have *I* annotations. Let the objects $o_1$ and $o_2$ have *I2* identical annotations. All objects $o_1$ … $o_n$ are presented to MULIC according to our ordering by decreasing frequency of annotations. We want to show that when $o_1$ and $o_2$ are presented to MULIC, at least *I2* annotations will match the closest cluster mode. There are generally 3 cases to consider: **a.** $o_3$…$o_n$ have the same set of *I2* identical annotations as $o_1$ and $o_2$. **b.** $o_3$…$o_n$ *do not* have the same set of *I2* identical annotations as $o_1$ and $o_2$ and $o_3$…$o_n$ are presented to MULIC *after* $o_1$ and $o_2$. **c.** $o_3$…$o_n$ *do not* have the same set of *I2* identical annotations as $o_1$ and $o_2$ and $o_3$…$o_n$ are presented to MULIC *before* $o_1$ and $o_2$.

In case **a**, $o_3$…$o_n$ are likely to also have another subset of identical annotations in *I-I2*. In this case, they will be presented *before* $o_1$ and $o_2$ to MULIC, so when $o_1$ and $o_2$ are presented the mode will contain all *I2* identical annotations and perhaps some more, so that $o_1$ and $o_2$ will have a similarity to the mode >= *I2*. In case **b**, $o_3$…$o_n$ are presented to MULIC *after* $o_1$ and $o_2$, so when $o_1$ and $o_2$ are presented the mode will contain all *I2* identical annotations so that $o_1$ and $o_2$ will have a similarity to the mode *equal to I2*. In case **c**, $o_3$…$o_n$ are presented to MULIC *before* $o_1$ and $o_2$, which means that they contain another subset of identical annotations *I3<=I-I2,* many of which are likely to match annotations in $o_1$ or $o_2$ (since they belong in the same class) so that $o_1$ and $o_2$ will have a similarity to the mode *near I2+I3*. Thus, when $o_1$ and $o_2$ are presented to MULIC the closest cluster mode will usually contain such values that the similarities of $o_1$ and $o_2$ to the mode will be near or at least as high as *I2*.



```
main() {
     object *p;      //linked list
     read_objects_into(p);
     order_items_in(p);
     MULIC(p);
}

MULIC(object *p) {
        num_CAs_can_differ = 0;

        repeat {

              for (all items i in p, from highest-ordered to lowest-ordered) {
                   cluster c = NULL;
                   find i's closest cluster c, comparing i with all cluster modes;
                   if (c == NULL OR number of different categorical annotations
                       between i and c.mode > num_CAs_can_differ)
                       { insert i in a new cluster; }
                   else { insert i in c; update c.mode; }
              }

              for (all clusters ac)
                   if (ac.number_of_items() == 1) { remove ac; }
                   else { remove all items in ac from p; }

              if (in this loop 0 items were placed in clusters with size > 1)
                   num_CAs_can_differ++;

        }until (p == NULL OR num_CAs_can_differ > stop_threshold);
}
```

**Fig. 2 – The MULIC clustering algorithm.**

## 4.3 Merging Clusters to Reduce Their Number

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 process our tool merges pairs of clusters whose top layers' modes' dissimilarity is lower than the maximum layer depth of the two clusters. For this purpose our tool 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 HA Indexes, as described later. This process takes little time:

```
for (c = first cluster to last cluster)
   for (d = c+1 to last cluster)
      if (dissimilarity(c->mode,d->mode) <
max(deepest_layer(c), deepest_layer (d)))
          merge(c, d);
```

## 4.4 Optimized version of MULIC

We developed an optimized version of MULIC for runtime purposes. The optimized version increases the similarity criterion *num_CAs_can_differ* by 3 or 5 at a time, while the non-optimized version increases it by 1 at a time. Sometimes, though not always, there is a slight loss in accuracy when increasing the criterion by 3 or 5 at a time; but the runtime is significantly better, as discussed in Section 5.

## 4.5 Dealing with outliers

MULIC can eventually put all the objects in clusters. When *num_CAs_can_differ* = *I*, 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_CAs_can_differ* – represented as *stop_threshold* in Figure 2 - so when this value is reached any remaining objects are not classified and are treated as outliers.

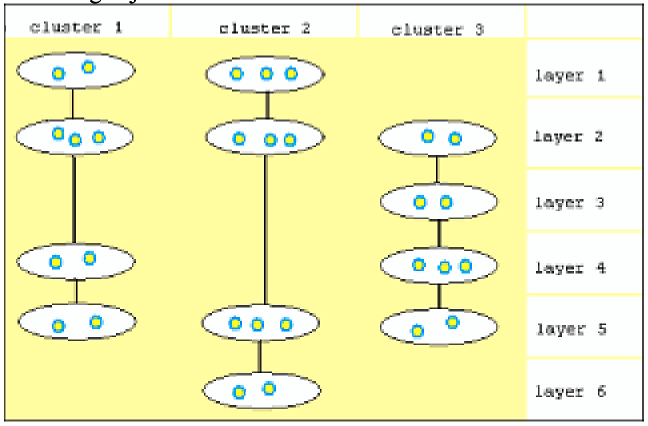


**Fig. 3 – Typical MULIC results. Each cluster consists of one or more different layers representing different coherences and similarities of the objects attached to the cluster.**

## 5. ACCURACY AND RUNTIME OF MULIC

We compared MULIC to other well-known algorithms - accuracy-wise and runtime-wise - on data sets of various sizes. We downloaded the test data sets from the Repository of Machine Learning Databases of the University of California, Irvine [17].

**Table 1 - Characteristics of the categorical test data sets.**

| Data Set | Objects | Attributes | Classes |
|---|---|---|---|
| Soybean-small | 47 | 35 | 4 |
| Zoo | 101 | 16 | 7 |
| Soybean-data | 307 | 35 | 19 |
| Soybean-test | 376 | 35 | 20 |
| Balance Scale | 625 | 4 | 3 |
| Contraceptive Method Choice | 1473 | 10 | 2 |
| Adult | 17884 | 14 | 2 |
| Adult-large | 32561 | 14 | 2 |

To compare the MULIC accuracy to that of other algorithms, we used *HA Indexes* [16]. Suppose we are given a set of *n* objects $S = \{O_1, \ldots, O_n\}$. Suppose that *U* is the criterion solution that represents the true partition known or believed to be present in *S*. *V* is the clustering result by some algorithm. Let *a* be the number of pairs of objects that are placed in the same class in *U* and in the same cluster in *V*, *b* be the number of pairs of objects in the same class in *U* but not in the same cluster in *V*, *c* be the number of pairs of objects in the same cluster in *V* but not in the same class in *U*, and *d* be the number of pairs of objects in different classes and different clusters in both partitions. Given these values, Hubert and Arabie defined the HA Index as below [16].

$$\text{HA Index} = \frac{a+d}{a+b+c+d}$$

We also used a measure of the *misclassification rate* of objects in clusters. Our misclassification rate is the *classes to clusters evaluation* that is used by the clustering algorithms of the WEKA package [18,20]. In this mode we first ignore the class attribute and generate the clustering. Then during the test phase we assign classes to the clusters, based on the majority value of the class attribute within each cluster. Then we compute the classification error, based on this assignment. To evaluate clustering using the classes to clusters approach we need to know the class values of the objects belonging to the clusters.

## 5.1 Accuracy Comparison of MULIC to k-Modes, AutoClass, ROCK, CLOPE

We compared the misclassification rates of MULIC to those of k-Modes [14] on data sets of various sizes. Our results are summarized in figure 4, showing that MULIC produced a significantly lower misclassification rate for all data sets. On some data sets of small sizes the misclassification rate was zero. We also compared the misclassification rate of MULIC to that of AutoClass [19] and ROCK [10] on data sets of various sizes. As shown in figure 4, the misclassification rate of MULIC was significantly lower than that of AutoClass and ROCK.

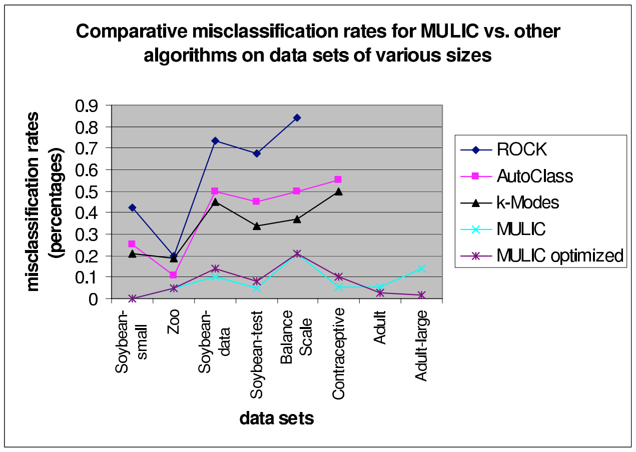


**Fig. 4 – MULIC vs. k-Modes, AutoClass and ROCK. The missing measurements for ROCK, AutoClass and k-Modes mean that the algorithm did not finish in a reasonable amount of time for the corresponding data sets.**

We developed a version of MULIC that was optimized for runtime. There was a slight tradeoff on accuracy for the optimized MULIC version, as shown in figure 4; the misclassification rate of the optimized MULIC was slightly higher than that of the non-optimized MULIC, on most but not all data sets. However, the misclassification rate of the optimized MULIC was still much lower than that of k-Modes, ROCK and AutoClass.

We also compared MULIC to all of the clustering algorithms that are contained in the WEKA data mining package [18,20]. Our results, shown in Figure 5, indicate that MULIC significantly outperformed all of the WEKA algorithms. These misclassification rates are output by WEKA automatically and represent the number of objects that were misclassified in the trial.

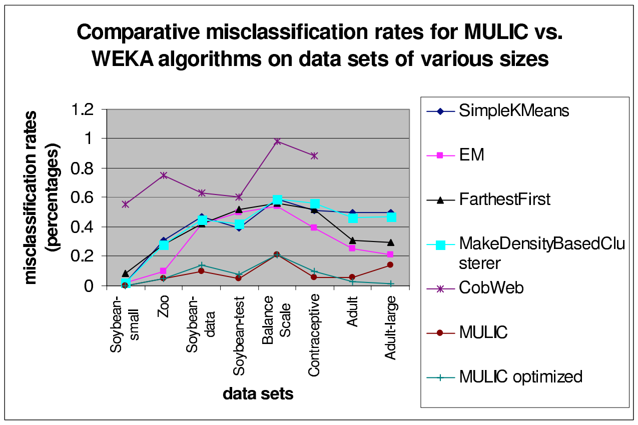


**Fig. 5 – MULIC vs. WEKA algorithms.**

We assessed the accuracy of MULIC using HA Indexes, which our software computes and outputs [16]. Our results, shown in figure 6, indicate that all HA indexes were very high - in some cases near 95%. Furthermore, we compared these to the HA Indexes of CLOPE [21] and k-Modes on the same data sets. For this purpose, we modified the CLOPE and k-Modes source code to output the HA Indexes. Figure 6 shows that MULIC's HA Indexes were higher than or comparable to CLOPE and k-Modes.

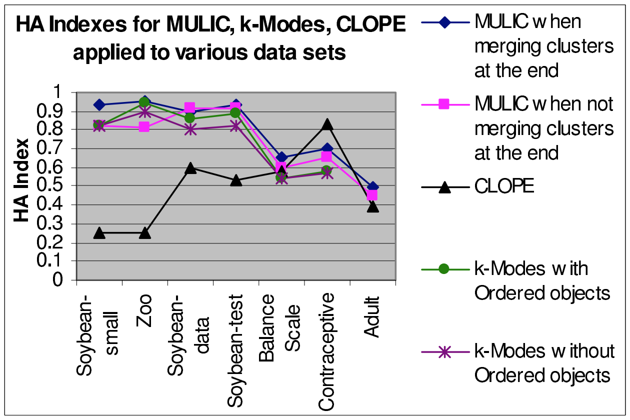


**Fig. 6 – HA Indexes for MULIC, k-Modes and CLOPE.**

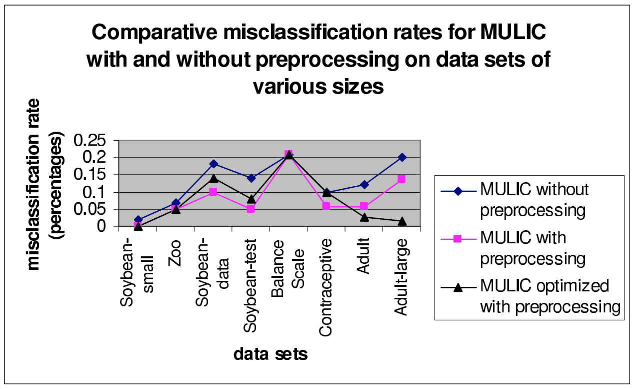


**Fig. 7 - The misclassification rates when preprocessing the data sets before clustering them with MULIC, as discussed in Section 4.1, versus not preprocessing the data sets beforehand.**

Figure 7 shows the MULIC misclassification rates when preprocessing the data sets before clustering by imposing an ordering on the objects based on the frequency of the annotations in the data set, as discussed in Section 4.1, versus not preprocessing the data sets beforehand. As shown, the results were more accurate when preprocessing the data sets.

We compared the results of different test runs using the *pessimistic error metric* introduced in Section 3. Table 2 shows the error metric for the optimized and non-optimized MULIC versions with preprocessing, MULIC without preprocessing and k-Modes, applied to various data sets.

**Table 2 – Pessimistic error metric comparison on 4 data sets.**

| Total objects in data set | 47 | 101 | 307 | 376 |
|---|---|---|---|---|
| MULIC without preprocessing | 78 | 119 | 898 | 1088 |
| MULIC with preprocessing | 73 | 110 | 812 | 966 |
| MULIC optimized with preprocessing | 73 | 110 | 856 | 995 |
| k-Modes | 89 | 135 | 991 | 1198 |

We performed paired two sided t-tests between pairs of distributions of different annotations amongst the clusters, to determine whether the distributions of two annotations are likely to have come from distributions with equal population means. As shown in table 3, the p-values are low meaning that for most pairs of annotation distributions it is unlikely that the distributions have the same mean and the difference in distributions observed is just a coincidence of random sampling; it is more likely that the distributions really have different means.

**Table 3 - Paired two sided t-test p-values for distributions amongst all clusters of pairs of annotation.**

| Pairs of annotation distributions amongst all clusters | Paired two sided t-test p-values for the pair of annotation distributions |
|---|---|
| '0 legs' and '5 legs' | 0.04 |
| '0 legs' and '6 legs' | 0.2 |
| '0 legs' and '7 legs' | 0.03 |
| '0 legs' and '8 legs' | 0.04 |
| '0 legs' and '4 legs' | 0.4 |

To justify the rationale for the top-most layer (e.g. *layer 0* or *1*) objects being the most influential objects in forming a cluster, we have computed the average annotation difference *a* between an object in the *top-most* layer of a cluster and all other objects in the cluster. Then, we compared *a* to the average annotation difference *b* between an object in the *lower* layer of the same cluster and all other objects in the cluster. We repeated this for several clusters in the results derived from clustering the 'zoo' data set. Table 4 shows that *a* is always lower than *b*, i.e. top layer objects are more representative of a cluster than low layer objects.

**Table 4 – Top layer objects represent a cluster better.**

| Clusters | Average difference *a* between an object in the *top-most* layer of a cluster and all other objects in the cluster | Average difference *b* between an object in the *lower* layer of a cluster and all other objects in the cluster |
|---|---|---|
| 1 | 1.8 | 2.7 |
| 2 | 1.1 | 2 |
| 3 | 0.6 | 1.5 |
| 4 | 0.7 | 1.3 |
| 5 | 1 | 1.7 |
| 6 | 0.8 | 1.3 |

## 5.2 Coherence of Clusters and Layers

Figures 8 and 9 show some of the results from clustering the 'zoo' data set with MULIC. Figure 8 shows that the resulting clusters and layers are very coherent. Most clusters have a very coherent layer 1 and layer 2. As layers decrease in a cluster, the layers' coherences also decrease. The overall cluster has a lower coherence than the layer 1 coherence. Figure 9 shows that most clusters have uniform annotation values for "number of legs" occurring at index 13 of the objects.

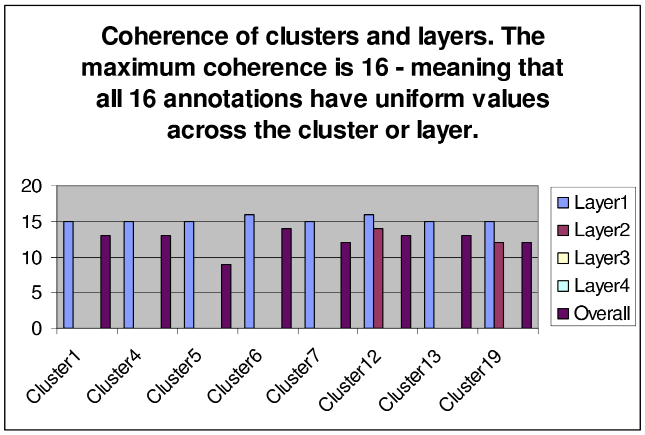


**Fig. 8 - The coherence of each layer in each cluster derived from clustering the 'zoo' data set, as well as the coherence of each overall cluster. The coherence is measured by counting how many annotations from the 16 annotations total have uniform values across all objects in the layer or cluster.**

A major advantage of MULIC is that in some data sets such as 'zoo', it was able to identify subclusters of extremely similar animals that had not been pointed out by the developers of the data set. For the 'zoo' data set, the animals 'porpoise', 'dolphin', 'sealion' and 'seal' were clustered together by MULIC in a cluster of their own, even though the developers of the data set had classified them as belonging to a much larger class consisting of 41 animals. *Under this observation, we do not view the merging of clusters as described in Section 4.3 as a crucial step for improving the results.*

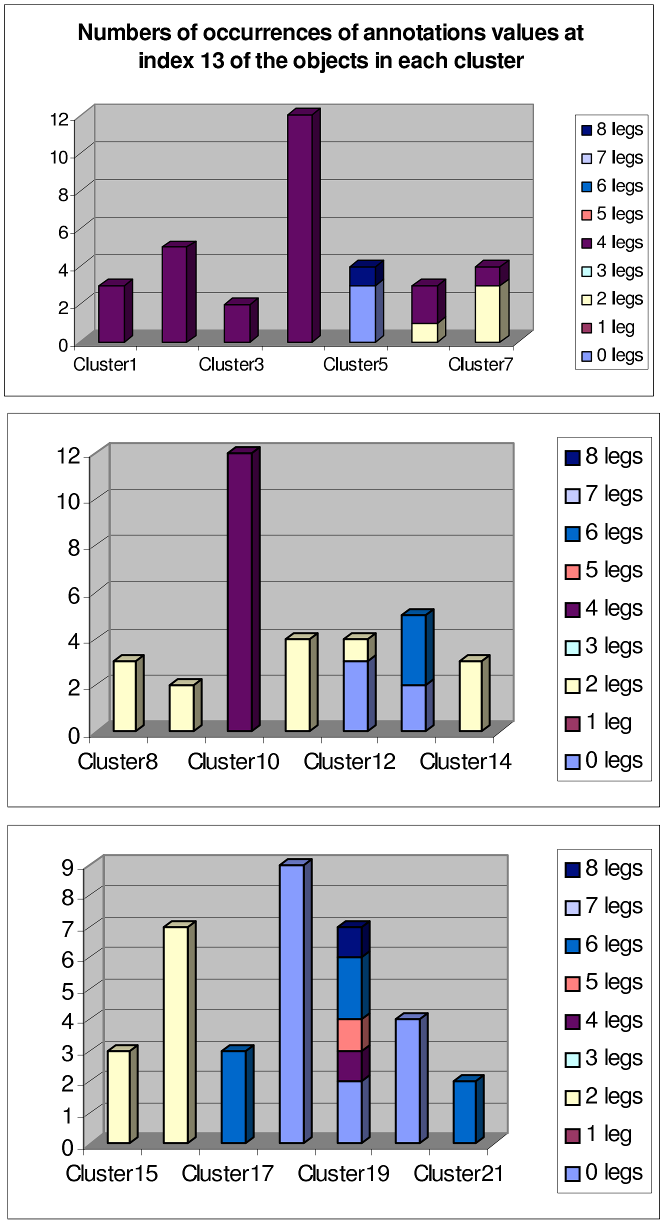


**Fig. 9a,b,c – Numbers of occurrences of annotation values for "number of legs" at index 13 of the objects in each cluster derived from clustering the 'zoo' data set.**

## 5.3 Runtime Comparison of MULIC to k-Modes, AutoClass, ROCK and Scalability

We compared the runtime of MULIC to that of k-Modes [14], ROCK [10] and AUTOCLASS [19] for data sets of various sizes. The runtime of MULIC was significantly better than ROCK and AUTOCLASS, as shown in figure 10. The runtimes of k-Modes and MULIC were comparable in most cases, as shown in figure 10. We also developed a version of MULIC that was optimized for runtime; the runtime of the optimized version of MULIC surpasses that of k-Modes, as shown in figure 10. There was a slight tradeoff on accuracy for the optimized MULIC version, as discussed in Section 5.1. However, the misclassification rate of the optimized MULIC was still much lower than that of k-Modes.

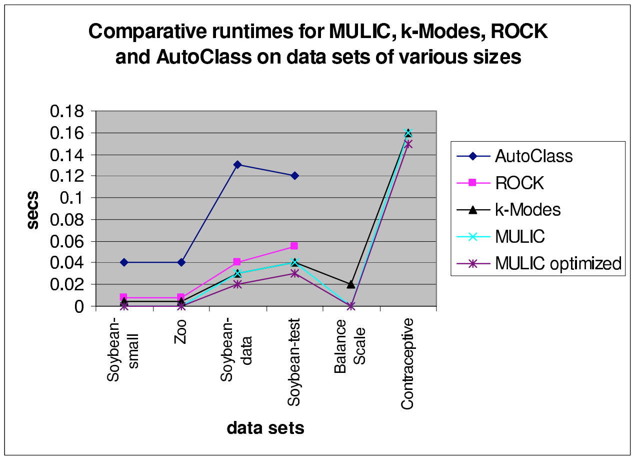


**Fig. 10 – Comparative runtimes for MULIC, k-Modes, ROCK and AutoClass on data sets of various sizes. The missing runtimes for AutoClass and ROCK mean that the algorithm did not finish in a reasonable amount of time for the corresponding data sets.**

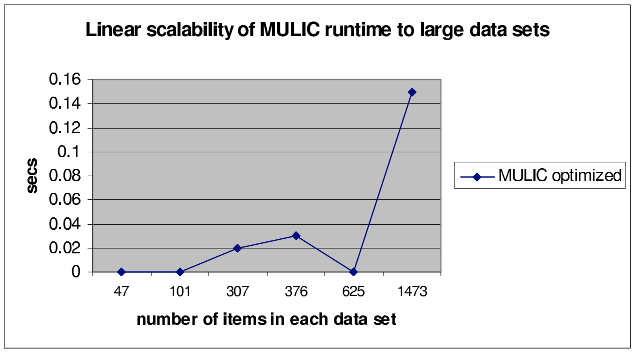


**Fig. 11 – As MULIC is applied to data sets of increasing sizes, its runtime increases linearly.**

Finally, we tested MULIC on data sets of large sizes, to show that the MULIC runtime scales up linearly with increasing data set size. Figure 11 shows the time in seconds that MULIC took to finish, for data sets of sizes varying from 47 objects to 1,473 objects.

## 5.4 Computational Complexity Analysis

The computational complexity of MULIC is *O(tn)*. This is comparable to the complexity of k-Modes [14]. Below we give a high-level and a more detailed analysis of the complexity of MULIC.

- *t*=number of categorical annotations (CAs) in each object.
- *n*=number of objects.
- Generally, t << n because the number of objects that we want to cluster will be much higher than the dimensionality of each object.
- *num_CAs_can_differ* is the similarity criterion for assessing if an object can be placed in the closest cluster for an iteration or not, based on how many CAs differ between the object and the cluster's mode. num_CAs_can_differ is incremented gradually throughout the iterations, to make the criterion more lenient.

In the worst case, objects in the data set to be clustered will be very dissimilar and MULIC will have difficulty allocating them to clusters. In this case, at most one object may be allocated to a cluster at an iteration and there will be many iterations where no objects are placed in clusters. In the worst case, before all n objects have been placed in clusters num_CAs_can_differ will equal t-1 and t; then MULIC will have no choice but to allocate all unclassified objects to clusters because the similarity criterion (num_CAs_can_differ) for classifying objects will have become very lenient.

We have calculated the complexity for the worst case, where from num_CAs_can_differ=0 to t-1 no objects are allocated to clusters. At each iteration all n objects in the set are considered but an unsuccessful attempt is made to allocate each object to a cluster.

When num_CAs_can_differ=t, then all n objects become allocated to clusters. However, they are placed one at each iteration. For example, one object is placed at iteration *k*, another is placed at iteration *k+1* and so on. At each iteration, we do not consider the objects that have already been allocated, only those that do not belong yet in clusters.

Thus, the complexity is:

$$O(\sum_{i=0}^{n}(n-i)+(t-1)n) = O(n+(t-1)n) = O(tn).$$

Here is a more detailed explanation of how to compute the worst case time complexity: there will be a finite number of iterations at which no object is classified in a cluster and then there will be a finite number of iterations at which one object is placed in a cluster for each iteration. So to calculate the complexity, we need to find how many iterations may exist at which no object is placed in a cluster and the cost of each such iteration and how many iterations may exist at which a single object is placed in a cluster and the cost of each such iteration.

If no object is placed in a cluster at an iteration, then num_CAs_can_differ will be increased so that an object might be placed at the next iteration. There can be at most t-1 iterations at which nothing is placed in clusters, because num_CAs_can_differ will reach t after t-1 such iterations and then some objects will have to be placed in clusters - in the worst case at the rate of 1 object per iteration. Each such iteration at which nothing is placed in clusters has cost n because all n objects have to be considered in an attempt to place each object in a cluster; thus, the cost of this step will be $O((t-1)n)$.

Then, in the worst case, all n objects will be placed in clusters one at each iteration without iterations at which none is placed - since num_CAs_can_differ = t, even if all CAs differ they will still be placed somewhere. So there will be a series of n iterations at which one object is placed in a cluster at each iteration. Each iteration will be less costly than the previous iteration, because at each iteration we do not consider the objects that have already been placed, only the objects that do not belong yet in clusters; thus, the cost of this step will be $O(\sum_{i=0}^{n}(n-i))$.

## 6. DISCUSSION

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

We have shown that the notion of using a 'mode' to summarize the contents of a cluster benefits from a special preprocessing and ordering on the input objects. A mode will be more beneficial for the correctness of the clustering results if the more frequently occurring annotations in the data set are presented to the process before the less frequently occurring annotations.

Future work will include extending MULIC to cluster mixed categorical and numerical data sets [1]. Furthermore, we would like to incorporate in the clustering process confidence values on the categorical annotations, indicating our certainty that the annotations are correct. We have described the incorporation of confidence values in the clustering process in [2].

## 7. CONCLUSIONS

We designed and implemented a clustering algorithm, called *MULIC*, that deals with many of the problems posed by k-Modes. Specifically, MULIC 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, MULIC finds the clusters that seem to exist naturally in the data set and forms as many clusters as required. For each cluster, MULIC 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 MULIC relaxes its criteria, forming layers that may be less coherent than the previous layers. Finally, MULIC imposes an ordering on the objects presented to the algorithm, thus ensuring that the results will always be the same. We have tested MULIC for accuracy on several data sets for which the correct result was known. On all of these data sets the MULIC misclassification rate was much lower than the misclassification rates of k-Modes, ROCK, AutoClass and the WEKA clustering algorithms. Finally, we compared the runtime of MULIC to that of k-Modes, ROCK, AutoClass and other algorithms showing that MULIC was at least as fast or faster.

## 8. 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. (2003) Significance Metrics for Clusters of Mixed Numerical and Categorical Yeast Data. Technical Report # CS-2003-12. Department of Computer Science, York University.
[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] D. Barbara, Y. Li, J. Couto. COOLCAT: an entropy-based algorithm for categorical clustering. In Proc of CIKM'02, pp. 582-589, 2002.
[5] Fasulo D. (1999) An Analysis of Recent Work on Clustering Algorithms, Technical Report # 01-03-02, Department of Computer Science & Engineering, University of Washington.
[6] V. Ganti, J. Gehrke, R. Ramakrishnan. CACTUS-clustering categorical data using summaries. In Proc of KDD '99, pp. 73-83.
[7] D. Gibson, J. Kleiberg, P. Raghavan. Clustering categorical data: an approach based on dynamic systems. In Proc of VLDB'98, pp. 311-323, 1998.
[8] Goebel, M. & Gruenwald, Le (1999). A survey of data mining and knowledge discovery software tools. ACM SIGKDD Explorations 1, 20-33.
[9] Grambeier J., Rudolph A. (2002) Techniques of Cluster Algorithms in Data Mining. Data Mining and Knowledge Discovery 6: 303-360.
[10] Guha S., Rastogi R., Shim K. (2000). ROCK: A Robust Clustering Algorithm for Categorical Attributes. Information Systems 25(5): 345-366.
[11] Hartigan, J. A. (1975) Clustering algorithms. (John Wiley and Sons, New York, 1975).
[12] Z. He, X. Xu, S. Deng: Squeezer: an efficient algorithm for clustering categorical data. Journal of Computer Science & Technology, 2002, 17(5): 611-624.
[13] Huang, Z. (1997) Clustering Large Data Sets with Mixed Numeric and Categorical Values. Knowledge discovery and data mining: techniques and applications. World Scientific.
[14] 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.
[15] Z. Huang, M.K.Ng. (1999). A fuzzy k-modes algorithm for clustering categorical data. IEEE Transaction on Fuzzy Systems, 1999, 7(4): 446-452.
[16] L. Hubert and P. Arabie, "Comparing partitions", Journal of Classification, 193–218, 1985.
[17] C.J.Mertz, P.Merphy. UCI Repository of Machine Learning Databases, 1996. (http://www.ics.uci.edu/~mlearn/MLRRepositoy.html).
[18] P. Reutemann, B. Pfahringer, E. Frank. (2004) Proper: A Toolbox for Learning from Relational Data with Propositional and Multi-Instance Learners. 17th Australian Joint Conference on Artificial Intelligence (AI2004). Springer-Verlag
[19] Stutz J. and Cheeseman P. (1995) Bayesian Classification (AutoClass): Theory and results. Advances in Knowledge Discovery and Data Mining, 153-180, Menlo Park, CA, AAAI Press.
[20] Ian H. Witten and Eibe Frank. "Data Mining: Practical machine learning tools with Java implementations". Morgan Kaufmann, San Francisco, 2000.
[21] Y. Yang, S. Guan, J. You. CLOPE: a fast and effective clustering algorithm for transactional data. In Proc of KDD'02, pp. 682-687, 2002.