# An Improved Genetic Clustering Algorithm for Categorical Data

Hongwu Qin, Xiuqin Ma, Tutut Herawan, and Jasni Mohamad Zain

Faculty of Computer Systems and Software Engineering
Universiti Malaysia Pahang
Lebuh Raya Tun Razak, Gambang 26300, Kuantan, Malaysia
{qhwump,xueener}@gmail.com, {tutut,jasni}@ump.edu.my

**Abstract.** Deng *et al*. [Deng, S., He, Z., Xu, X.: G-ANMI: A mutual information based genetic clustering algorithm for categorical data, Knowledge-Based Systems 23, 144--149(2010)] proposed a mutual information based genetic clustering algorithm named G-ANMI for categorical data. While G-ANMI is superior or comparable to existing algorithms for clustering categorical data in terms of clustering accuracy, it is very time-consuming due to the low efficiency of genetic algorithm (GA). In this paper, we propose a new initialization method for G-ANMI to improve its efficiency. Experimental results show that the new method greatly improves the efficiency of G-ANMI as well as produces higher clustering accuracy.

**Keywords:** Data mining, Clustering, Categorical data, Genetic algorithm.

## 1 Introduction

Clustering is an important data mining technique that groups together similar data objects. Most previous clustering algorithms focus on numerical data whose inherent geometric properties can be exploited naturally to define distance functions between objects. However, many fields, from statistics to psychology deal with categorical data. Unlike numerical data, categorical data cannot be naturally ordered. An example of categorical attribute is *color* whose values include *red*, *green*, *blue*, etc. Therefore, those clustering algorithms dealing with numerical data can not be used to cluster categorical data. Recently, the problem of clustering categorical data has received much attention [1-10].

Categorical data clustering has been defined as an optimization problem which aims to find an optimal partition of the objects according to an objective function [1-7]. Unfortunately, this optimization problem is NP-complete. Therefore most researchers resort to heuristic methods to solve it, such as ROCK [1], k-modes [2], COOLCAT [3], and k-ANMI [4]. However, these algorithms tend to find local optimal partition. Recently, some genetic clustering algorithms have been proposed to find globally optimal or near-optimal partition, such as ALG-RAND [6] and G-ANMI [7] algorithms. In the performance comparison conducted in [7], it has shown that

T. Washio and J. Luo (Eds.): PAKDD 2012 Workshops, LNAI 7769, pp. 100–111, 2013.

G-ANMI is superior or comparable to ALG-RAND as well as other existing algorithms for clustering categorical data in terms of clustering accuracy. However, G-ANMI is very time-consuming. For instance, it takes G-ANMI 20759 seconds to mine 2 clusters from Mushroom dataset [11] with 8124 objects. Thus, it is necessary to improve its efficiency before it can be widely used in practice.

The low efficiency of G-ANMI is mainly caused by GA [8] which needs a lot of iterations to find the optimal solution. Given a population size, the efficiency of G-ANMI is dominated by the number of iterations. Hence, we have to reduce the number of iterations to improve the efficiency of G-ANMI. In a categorical data set, each attribute defines a partition of the objects. The aim of G-ANMI is to find a k-partition (k is the desired number of clusters) that shares the most information with the partitions defined by attributes (attributes partitions for short). In other words, G-ANMI tries to find a k-partition that is the closest to the attributes partitions. However, G-ANMI algorithm starts with a population of randomly generated k-partitions of objects. These randomly generated k-partitions are far from the attributes partitions when we process a larger data set. The farther these partitions are from the attributes partitions, the more iteration G-ANMI needs to reach the optimal k-partition. Hence, it is possible to reduce the number of iterations of G-ANMI by giving some better initial k-partitions which are closer to the attributes partitions in comparison with those randomly generated k-partitions.

In this paper, we propose a new initialization method for G-ANMI, in which some equivalence classes (the set of objects which has the same value on an attribute) in attributes partitions are directly integrated into the initial k-partitions. The initial k-partitions obtained by using the new method are closer to the attributes partitions in comparison with those randomly generated k-partitions, especially when we process a larger data set. As a result, less number of iterations is needed to reach the optimal k-partition. Experimental results show that the new method greatly improves the efficiency of G-ANMI, as well as produces higher clustering accuracy. The rest of the paper is organized as follows. Section 2 briefly introduces G-ANMI algorithm. Section 3 presents the new initialization method. Section 4 presents experimental results on UCI benchmark data sets. Finally, Section 5 presents conclusions and future work.

## 2 G-ANMI

G-ANMI employs basic GA to implement categorical data clustering, which works in the same way as the one used in ALG-RAND [6].

G-ANMI starts with a population of randomly generated partitions of objects, which are encoded as chromosomes. If the desired number of clusters is set to $k$, then each chromosome is encoded as a $k$-partition of objects. Suppose the integers between interval [0, $k$-1] are used as class identifier, a chromosome will be a string of integers which are between interval [0, $k$-1]. For example, suppose the number of objects is 20, and $k$ is 4, a randomly generated chromosome is as follows

1 0 2 0 1 0 3 2 3 1 0 1 2 0 3 2 0 1 1 2

Then, G-ANMI uses the average normalized mutual information (ANMI) to evaluate the fitness of each chromosome in the current population. Given a set of $r$ partitions

defined by attributes: $\Lambda = \{\lambda^{(q)} \mid q \in \{1,2,...,r\}\}$ and a partition $\overline{\lambda}$, the average normalized mutual information (ANMI) between $\Lambda$ and $\overline{\lambda}$ is defined as follows:

$$\phi^{(ANMI)}(\Lambda, \overline{\lambda}) = \frac{1}{r}\sum_{q=1}^{r}\phi^{(NMI)}(\overline{\lambda}, \lambda^{(q)}) \tag{1}$$

where $\phi^{(NMI)}(\overline{\lambda}, \lambda^{(q)})$ denotes the normalize mutual information between $\lambda^{(q)}$ and $\overline{\lambda}$. Without loss of generality, normalized mutual information between two partitions $\lambda^{(a)}$ and $\lambda^{(b)}$ is computed as follows:

$$\phi^{(NMI)}(\lambda^{(a)}, \lambda^{(b)}) = \frac{2}{n}\sum_{h=1}^{k^{(a)}}\sum_{g=1}^{k^{(b)}} n_g^{(h)} \log_{k^{(a)} * k^{(b)}}\left(\frac{n_g^{(h)} n}{n^{(h)} n_g}\right) \tag{2}$$

where $k^{(a)}$ and $k^{(b)}$ are the number of clusters in partition $\lambda^{(a)}$ and $\lambda^{(b)}$, respectively. $n^{(h)}$ denotes the size of cluster $C_h$ in partition $\lambda^{(a)}$, $n_g$ denotes the size of cluster $C_g$ in partition $\lambda^{(b)}$, $n_g^{(h)}$ denotes the number of shared objects between $C_h$ and $C_g$.

According to the fitness value, genetic evolution repeatedly changes the chromosomes in the current population to generate a new population. It is expected that chromosomes could be increasingly closer to the optimal partition with largest ANMI. Genetic procedure will halt when the best fitness in the current population is greater than the user-specified fitness threshold or there has been no relative improvement on best fitness after some consecutive iterations.

## 3 New Initialization Method

The basic idea of the new initialization method is that integrating some equivalence classes of the partitions defined by attributes into the generation of initial partitions. Two cases are considered:

**i.** If the population size *P* is greater than or equal to the number of attributes *M*, then the algorithm generates first *M* chromosomes from the *M* attributes partitions, and generates other *P*-*M* chromosomes randomly.

**ii.** If the population size *P* is less than the number of attributes *M*, then the algorithm generates *P* chromosomes from the first *P* attributes partitions.

Generating chromosomes from the attributes partitions is implemented by a one-one way, namely one chromosome is generated by one partition. Generating a chromosome from a partition means taking some equivalence classes of the partition as the part of the chromosome. How many equivalence classes should we take depends on the number of equivalence classes (*Nec*) in the partition and the specified number of clusters *k*. Different strategies are employed when the number of equivalence classes in the partition is greater than, less than and equals to the specified number of clusters, respectively. The details are described in Fig .1.

**Begin**

**For** each partition *Par*
**if** *Nec* in *Par* equals *k*
Copy *Par* to the corresponding chromosome *Chrom*
**else**
**if** *Nec* is greater than *k*
Copy first *k* equivalence classes of *Par* to the same locations in *Chrom*.
Generate a random number between [0, *k*-1] for each of the remaining locations in *Chrom*.
**else**
Find a highest *H* which satisfies the following inequation
*N* - Sum >= *k*-*H*-1
//where *N* is the length of a chromosome, Sum is the summation
//of the size of first *H*+1 equivalence classes of *Par*.
Copy first *H*+1 equivalence classes of *Par* to the same locations in *Chrom*.
Generate a random number between [*H*+1, *k*-1] for each of the remaining locations in *Chrom*.
**End**.

**Fig. 1.** The procedure of generating a chromosome from a partition

Note that the purpose of inequation *N* - Sum >= *k*-*H*-1 is to ensure each number between interval [*H*+1, *k*-1] appears at least once in *Chrom* when generating a random number for each of the remaining locations in *Chrom*.

Next, we present an illustrative example of the new initialization method. For the comparison purpose, the G-ANMI algorithm with new initialization method is named improved G-ANMI (IG-ANMI).

**Example 1.** Suppose there is a data set with ten objects ($O_1$, $O_2$, …, $O_{10}$) and four attributes ($A_0$, …, $A_3$). Table 1 shows the partitions defined by the four attributes. The numbers 0, 1, 2, and 3 denote different equivalence classes (categories) in the partitions. We use the algorithms IG-ANMI and G-ANMI to cluster the objects, respectively. The parameter setting includes: the number of clusters $k$=3, the population size $P$=10, crossover rate=0.8, mutation rate=0.1, random seed=1, and the number of consecutive iterations without improvement=100.

Since the population size $P$ is greater than the number of attributes, we generate first four chromosomes by using attributes partitions and generate remaining six chromosomes randomly. The attributes partitions are named *Par*[*i*], *i*=0, 1, 2, 3. The chromosomes are named *C*[*j*], *j*=0, 1, …, 9. The numbers of equivalence classes in *Par*[0] and *Par*[2] equal the specified number of clusters *k*, so we directly copy *Par*[0] and *Par*[2] to *C*[0] and *C*[2], respectively. The number of equivalence classes in *Par*[1] is less than *k*. According to the algorithm shown in Figure 1, we first seek an appropriate number *H*. In this example, there is only possible value for *H*, namely zero. Zero satisfies *N* - Sum >= *k*-*H*-1, thus *H* gets the value zero. Next, we copy the

**Table 1.** The partitions defined by four attributes

| $U$ | $A_0$ | $A_1$ | $A_2$ | $A_3$ |
|---|---|---|---|---|
| $O_1$ | 0 | 0 | 0 | 0 |
| $O_2$ | 1 | 0 | 1 | 1 |
| $O_3$ | 0 | 1 | 0 | 0 |
| $O_4$ | 0 | 0 | 0 | 0 |
| $O_5$ | 1 | 1 | 2 | 2 |
| $O_6$ | 1 | 1 | 1 | 2 |
| $O_7$ | 2 | 0 | 2 | 3 |
| $O_8$ | 2 | 1 | 2 | 1 |
| $O_9$ | 1 | 1 | 1 | 2 |
| $O_{10}$ | 2 | 1 | 2 | 3 |

first equivalence class to the corresponding location in $C[1]$. Table 2 shows the status of $C[1]$ after copying the first equivalence class. There are still six locations need to be filled in $C[1]$. We generate a random number between interval [1, 2] for each of the six locations.

**Table 2.** The status of $C[1]$ after copying the first equivalence class

| Location | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| $C[1]$ | 0 | 0 | | 0 | | | 0 | | | |

The number of equivalence classes in $Par[3]$ is greater than $k$. According to the algorithm shown in Figure 1, we copy first three equivalence classes of $Par[3]$ to the corresponding location in $C[3]$. Table 3 shows the status of $C[3]$ after copying first three equivalence classes. There are still two locations need to be filled in $C[3]$. We generate a random number between interval [0, 2] for each of the two locations.

**Table 3.** The status of $C[3]$ after copying first three equivalence classes

| Location | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| $C[3]$ | 0 | 1 | 0 | 0 | 2 | 2 | | 1 | 2 | |

Following the method, the remaining six chromosomes $C[4]$, … $C[9]$ are randomly generated. At the end, ten chromosomes are obtained and summarized in Table 4. The numbers in bold style are randomly generated.

Note that the equivalence classes in each of the first four chromosomes are labeled by order 0, 1, 2. However, the equivalence classes in each of other six chromosomes are labeled unorderly. Actually, the numbers 0, 1, 2 in the partitions or chromosomes only denote different categories rather than order. That means the order of the labels doesn't affect the computation of fitness of a chromosome. Even if we change the order of the labels in some chromosomes, their fitness values keep invariable. For instance, we can change $C[0]$ from {0, 1, 0, 0, 1, 1, 2, 2, 1, 2} to {1, 2, 1, 1, 2, 2, 0, 0, 2, 0}, change $C[4]$ from {1, 1, 1, 2, 1, 1, 0, 1, 0, 0} to {0, 0, 0, 1, 0, 0, 2, 0, 2, 2}, and so on.

**Table 4.** Ten chromosomes generated by the new initialization method

| Location | *C*[0] | *C*[1] | *C*[2] | *C*[3] | *C*[4] | *C*[5] | *C*[6] | *C*[7] | *C*[8] | *C*[9] |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | **1** | **2** | **1** | **2** | **1** | **1** |
| 1 | 1 | 0 | 1 | 1 | **1** | **0** | **1** | **1** | **1** | **2** |
| 2 | 0 | **1** | 0 | 0 | **1** | **2** | **0** | **0** | **0** | **0** |
| 3 | 0 | 0 | 0 | 0 | **2** | **1** | **1** | **1** | **1** | **0** |
| 4 | 1 | **1** | 2 | 2 | **1** | **2** | **1** | **1** | **0** | **2** |
| 5 | 1 | **1** | 1 | 2 | **1** | **2** | **1** | **1** | **0** | **0** |
| 6 | 2 | 0 | 2 | **0** | **0** | **2** | **2** | **0** | **1** | **0** |
| 7 | 2 | **1** | 2 | 1 | **1** | **2** | **2** | **0** | **0** | **1** |
| 8 | 1 | **1** | 1 | 2 | **0** | **0** | **1** | **2** | **2** | **2** |
| 9 | 2 | **2** | 2 | **1** | **0** | **0** | **2** | **1** | **0** | **0** |

After the initialization, the next step of IG-ANMI is to calculate the fitness of each chromosome. According to the Eq. (1), we obtain the fitness of chromosomes as is shown in Table 5.

**Table 5.** The fitness of initial chromosomes of IG-ANMI

| Chromosomes | fitness value | average |
|---|---|---|
| *C*[0] | 0.654067 | 0.53927 |
| *C*[1] | 0.361562 | |
| *C*[2] | 0.615644 | |
| *C*[3] | 0.525807 | |
| *C*[4] | 0.252361 | 0.265807 |
| *C*[5] | 0.196573 | |
| *C*[6] | 0.403407 | |
| *C*[7] | 0.166014 | |
| *C*[8] | 0.323139 | |
| *C*[9] | 0.253349 | |
| average | 0.375192 | |

It can be seen from Table 5 that the average fitness value of first four chromosomes is higher than that of other six chromosomes, which indicates that the chromosomes generated from the attributes partitions are closer to the optimal partition than that generated randomly. With these fitness values, the algorithm IG-ANMI generates new population and goes into the next iteration. Since there has been no relative improvement on best fitness during 100 consecutive iterations, the algorithm IG-ANMI ends after the $100^{th}$ iteration. Finally, we get the optimal 3-partition {0, 1, 0, 0, 1, 1, 2, 2, 1, 2}.

We use G-ANMI algorithm to cluster the same data set below. Firstly, G-ANMI randomly generates *P* chromosomes as is shown in Table 6.

Table 7 shows the fitness values of the chromosomes in the initial population. Obviously, the average fitness as well as the best fitness of the chromosomes is less than that in the initial population generated by algorithm IG-ANMI. After 27 iterations, the best fitness reaches 0.654067, which equals to the best fitness of the initial population generated by algorithm IG-ANMI. Since there has been no relative improvement on

best fitness during the subsequent 99 consecutive iterations, the algorithm G-ANMI ends after the 127th iterations. G-ANMI needs 27 more iterations than IG-ANMI due to the randomly generated initial population.

**Table 6.** Ten chromosomes generated by the initialization method of G-ANMI

| Location | *C*[0] | *C*[1] | *C*[2] | *C*[3] | *C*[4] | *C*[5] | *C*[6] | *C*[7] | *C*[8] | *C*[9] |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 2 | 0 | 0 | 0 | 0 | 0 | 2 | 2 |
| 1 | 1 | 2 | 1 | 1 | 2 | 1 | 1 | 0 | 1 | 1 |
| 2 | 2 | 1 | 2 | 1 | 2 | 1 | 0 | 2 | 1 | 0 |
| 3 | 1 | 1 | 2 | 1 | 0 | 1 | 0 | 0 | 2 | 1 |
| 4 | 1 | 0 | 2 | 2 | 0 | 0 | 1 | 0 | 2 | 2 |
| 5 | 1 | 1 | 2 | 2 | 0 | 0 | 0 | 1 | 1 | 1 |
| 6 | 0 | 0 | 0 | 1 | 2 | 2 | 2 | 2 | 2 | 2 |
| 7 | 1 | 0 | 0 | 2 | 0 | 1 | 0 | 0 | 0 | 1 |
| 8 | 1 | 2 | 1 | 0 | 2 | 1 | 2 | 1 | 1 | 2 |
| 9 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |

**Table 7.** The fitness values of the chromosomes in the initial population of G-ANMI

| Chromosomes | fitness value |
|---|---|
| *C*[0] | 0.211672 |
| *C*[1] | 0.469059 |
| *C*[2] | 0.338877 |
| *C*[3] | 0.227614 |
| *C*[4] | 0.139958 |
| *C*[5] | 0.181013 |
| *C*[6] | 0.216344 |
| *C*[7] | 0.263182 |
| *C*[8] | 0.377376 |
| *C*[9] | 0.166627 |
| average | 0.259172 |

## 4 Experimental Results

A series of experiments are conducted to evaluate the clustering efficiency and clustering performance of IG-ANMI. They are described below.

### 4.1 Experiments Design

We aim to evaluate the influence of new initialization method on G-ANMI algorithm. Therefore, the experimental studies are devoted to the comparison between G-ANMI and IG-ANMI. Four real-life datasets obtained from the UCI Machine Learning Repository [11] are used in the experiments, including Zoo, Congressional Votes (Votes for short), Wisconsin Breast Cancer (Breast Cancer for short), and Mushroom. The reason for choosing these four datasets is that they are also used in G-ANMI for evaluation. The information about the data sets is tabulated in Table 8.

**Table 8.** The information about the four data sets

| Data set | Number of objects | Number of Attributes | Number of classes |
|---|---|---|---|
| Zoo | 101 | 16 | 7 |
| Votes | 435 | 16 | 2 |
| Breast cancer | 699 | 9 | 2 |
| Mushroom | 8124 | 22 | 2 |

The parameters required by G-ANMI and IG-ANMI are set to be the same as in [7]. In addition, population size has a great effect on the quality of clustering in G-ANMI and IG-ANMI. In our experiments we vary the population size to perform the comparison between G-ANMI and IG-ANMI. For the Zoo, Votes, and Breast Cancer data sets, the population size vary from 50 to 500, for the Mushroom data set, the population size varies from 50 to 200.

All the programs are written in C language and compiled on the Borland C++ version 5.02. All experiments are conducted on a machine with Intel Core2 Duo CPU T7250 @ 2.00GHz, 1.99 GB of RAM, running Microsoft Windows Vista.

### 4.2 Efficiency Analysis

In our experiments, the running time of algorithms is used as the criteria for efficiency evaluation. Figs. 2-5 plot the running time of G-ANMI and IG-ANMI in seconds on four data sets when population size is increased. It can be seen that IG-ANMI takes less running time than G-ANMI except for on the Zoo data set when population size is 500. It is worth noting that there is a very large difference between G-ANMI and IG-ANMI on the Mushroom data set, which indicates IG-ANMI can save much time when larger data sets are processed. Table 9 shows the concrete values of numbers of iterations and running time of G-ANMI and IG-ANMI on the Mushroom data set. When the population size is set to 200, G-ANMI takes 190998.485 seconds (53 hours) while IG-ANMI only take 1351.594 seconds.

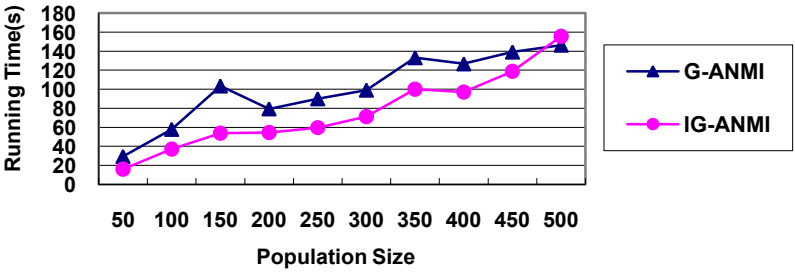


**Fig. 2.** Running [1]time vs. population size on the Zoo data set

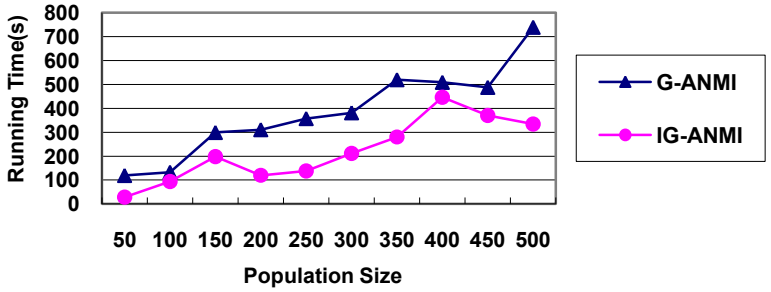


**Fig. 3.** Running time vs. population size on the Votes data set

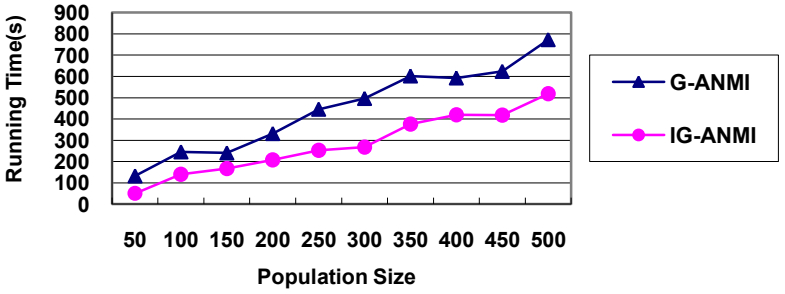


**Fig. 4.** Running time vs. population size on the Breast Cancer data set

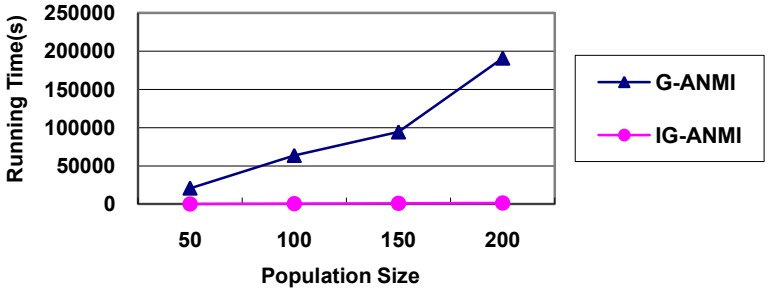


**Fig. 5.** Running time vs. population size on the Mushroom data set

**Table 9.** The numbers of iterations and running time of G-ANMI and IG-ANMI on the Mushroom data set

| Population Size | Number of iterations | | Running time (s) | |
|---|---|---|---|---|
| | G-ANMI | IG-ANMI | G-ANMI | IG-ANMI |
| 50 | 10845 | 100 | 20759.969 | 201.25 |
| 100 | 14453 | 145 | 63574.047 | 606.312 |
| 150 | 13944 | 144 | 94324.032 | 880.875 |
| 200 | 17916 | 158 | 190998.485 | 1351.594 |

**Table 10.** The clustering accuracies of G-ANMI and IG-ANMI on four data sets

| Data set | Algorithms | 50 | 100 | 150 | 200 | 250 | 300 | 350 | 400 | 450 | 500 | Average |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Zoo | G-ANMI | 0.832 | 0.851 | 0.911 | 0.881 | 0.861 | 0.891 | 0.931 | 0.812 | 0.851 | 0.921 | 0.874 |
| | IG-ANMI | 0.881 | 0.911 | 0.901 | 0.911 | 0.921 | 0.921 | 0.921 | 0.931 | 0.931 | 0.931 | 0.916 |
| Votes | G-ANMI | 0.874 | 0.862 | 0.876 | 0.864 | 0.880 | 0.874 | 0.869 | 0.864 | 0.876 | 0.874 | 0.871 |
| | IG-ANMI | 0.839 | 0.860 | 0.880 | 0.862 | 0.867 | 0.869 | 0.876 | 0.876 | 0.871 | 0.878 | 0.868 |
| Breast Cancer | G-ANMI | 0.924 | 0.962 | 0.960 | 0.963 | 0.972 | 0.975 | 0.975 | 0.975 | 0.977 | 0.978 | 0.966 |
| | IG-ANMI | 0.936 | 0.966 | 0.965 | 0.969 | 0.971 | 0.977 | 0.975 | 0.977 | 0.972 | 0.978 | 0.969 |
| Mushroom | G-ANMI | 0.547 | 0.568 | 0.546 | 0.538 | | | | | | | 0.55 |
| | IG-ANMI | 0.744 | 0.902 | 0.847 | 0.901 | | | | | | | 0.849 |

### 4.3 Performance Analysis

We use clustering accuracy to evaluate the performance of the IG-ANMI, which is one of the most widely used methods to evaluate the results of clustering algorithms. Given the true class labels and the required number of clusters, $k$, clustering accuracy is defined as $\frac{\sum_{i=1}^{k} a_i}{n}$, where $n$ is number of objects in the dataset and $a_i$ is the number of objects with the class label that dominates cluster $i$. A higher value of clustering accuracy indicates a better clustering result. The clustering accuracies of two algorithms on four data sets are summarized in Table 10. From the average accuracies, we can see that IG-ANMI has higher clustering accuracy on the Zoo, Breast Cancer, and Mushroom data sets. One exception is on the Votes data set, the clustering accuracy of G-ANMI is slightly higher than that of IG-ANMI. It is worth noting that IG-ANMI improves clustering accuracy greatly on the Mushroom data set.

## 5 Conclusions

In this paper, we propose a new initialization method for G-ANMI, namely integrating some equivalence classes of the attributes partitions into the generation of initial partitions. Experimental results on four real-life data sets show that the new method greatly improves the efficiency of G-ANMI, as well as produces higher clustering accuracy, especially on the larger data sets. The new initialization method could be more complicated so that producing better initial chromosomes. In the future work, we will develop other initialization methods to further improve the efficiency of G-ANMI.

**Acknowledgments.** This work was supported by PRGS under the Grant No. GRS100323, Universiti Malaysia Pahang, Malaysia.

## References

1. Guha, S., Rastogi, R., Shim, K.: ROCK: a robust clustering algorithm for categorical attributes. Information Systems 25(5), 345–366 (2000)
2. Huang, Z.: Extensions to the k-means algorithm for clustering large data sets with categorical values. Data Mining and Knowledge Discovery 2(3), 283–304 (1998)
3. Barbara, D., Li, Y., Couto, J.: COOLCAT: an entropy-based algorithm for categorical clustering. In: Proc. of CIKM 2002, pp. 582–589 (2002)
4. He, Z., Xu, X., Deng, S.: k-ANMI: a mutual information based clustering algorithm for categorical data. Information Fusion 9(2), 223–233 (2008)
5. He, Z., Xu, X., Deng, S.: A cluster ensemble method for clustering categorical data. Information Fusion 6(2), 143–151 (2005)
6. Cristofor, D., Simovici, D.: Finding median partitions using information-theoretical-based genetic algorithms. Journal of Universal Computer Science 8(2), 153–172 (2002)

7. Deng, S., He, Z., Xu, X.: G-ANMI: A mutual information based genetic clustering algorithm for categorical data. Knowledge-Based Systems 23, 144–149 (2010)
8. Holland, J.H.: Adaptation in Natural and Artificial Systems. MIT Press (1992)
9. Bai, L., Liang, J.Y., Dang, C.Y.: An initialization method to simultaneously find initial cluster and the number of clusters for clustering categorical data. Knowledge-Based Systems 24, 785–795 (2011)
10. Herawan, T., Deris, M.M., Abawajy, J.H.: A rough set approach for selecting clustering attribute. Knowledge-Based Systems 23, 220–231 (2010)
11. UCI Machine Learning Repository (2011), http://www.ics.uci.edu_/mlearn/MLRepository.html