Combining K-Means and a Genetic Algorithm through a Novel Arrangement of Genetic Operators for High Quality Clustering

Md Zahidul Islam, Vladimir Estivill-Castro, Md Anisur Rahman, and Terry Bossomaier.

***Abstract*—Knowledge discovery from data demands that it shall be the data themselves that reveal the groups (i.e. the data elements in each group) and the number of groups. For the ubiquitous task of clustering,** K-MEANS **is the most used algorithm applied in a broad range of areas to identify groups where intra-group distances are much smaller than inter-group distances. As a representative-based clustering approach,** K-MEANS **offers an extremely efficient gradient descent approach to the total squared error of representation; however, it not only demands the parameter k, but it also makes assumptions about the similarity of density among the clusters. Therefore, it is profoundly affected by noise. Perhaps more seriously, it can often be attracted to local optima despite its immersion in a multi-start scheme. We present an effective genetic algorithm that combines the capacity of genetic operators to conglomerate different solutions of the search space with the exploitation of the hill-climber. We advance a previous genetic-searching approach called** GENCLUST**, with the intervention of fast hill-climbing cycles of** K-MEANS **and obtain an algorithm that is faster than its predecessor and achieves clustering results of higher quality. We demonstrate this across a series of 18 commonly researched datasets.**

***Index Terms*—Clustering, Genetic Algorithm,** K-MEANS**, Data Mining, Cluster Evaluation.**

1. INTRODUCTION

Clustering groups the records of a dataset in such a way that similar records are grouped together in a cluster and dissimilar records are placed in different clusters. It has a broad range of applications in almost all areas including gene analysis (Brameier & Wiuf, 2007), medical imaging (Loai, Lin, & Li, 2008) and market analysis (Chan, Kwong, & Hu, 2012).

As a result, many clustering techniques have been proposed, out of which K-MEANS (Lloyd, 1982) is one of the most popular techniques (Wu et al., 2008). K-MEANS tops Expectation Maximization among clustering methods and the top 10 most used algorithms in data analysis (Wu et al., 2008). Although K-MEANS assumes that the data present clusters that are essentially well separated and spherical, it is regularly applied to situations for which perhaps there is little justification for this. For example, the construction of the vocabulary tree (Nistér & Stewénius, 2006) in sophisticated object recognition tasks from images by techniques such as SIFT. This is so because even if the data is not a mixture of multivariate normals, identifying the high-density regions represented by a set of centers and radii is useful information providing insights into the data.

Other strong reasons for adopting K-MEANS include its speed, its capacity to work by scanning the data sequentially, the possibility to stop the algorithm at any time and be provided with some answer, with also being able to resume K-MEANS with different subsets of the data or additional data records. Its simplicity (the centroid finding and centroid assignment are two simple phases of its core iterative loop) displays a linear-time complexity (that is, $O(n)$ time) for clustering a dataset with n records.

However, it has a number of drawbacks. Some of them derived from its inductive *least sum of squares* principle (Estivill-Castro, 2002), but some others are due to the algorithm's structure. First, it requires the number k of clusters as an input. In realistic settings, it can be extremely difficult for a user to guess the number of clusters in advance (Jain, 2010). Second, K-MEANS is sensitive to the initial seeds (i.e. cluster centers or genes) that it produces randomly (Arthur & Vassilvitskii, 2007). Bad initial seeds can easily lead K-MEANS to poor clustering results (Rahman & Islam, 2011). Third, since it is a simple hill-climber method for its objective function, it can get trapped at a local optimum and thus produce a poor clustering solution (Maulik & Bandyopadhyay, 2000). It assumes that all clusters have approximately equal density and similar spread; as a result it can be misguided by noise and outliers.

Hence, there is a huge demand for clustering techniques that are simple but free from the limitations of K-MEANS. Genetic algorithms have been used along with K-MEANS to avoid the requirement of the user input on k and improve the cluster quality by exploring good quality optima (Maulik & Bandyopadhyay, 2000; Bandyopadhyay & Maulik, 2002; Laszlo & Mukherjee, 2007; Liu, Wu, & Shen, 2011).

Many genetic algorithms select their initial population randomly (Liu et al., 2011; Chang, Zhang, & Zheng, 2009) which may have an adverse impact on final clustering results. A recent technique called GENCLUST uses an advanced approach in selecting its initial population which was shown (Rahman & Islam, 2014) to be useful in achieving better clustering results. Nevertheless, due to the complex initial population selection process, it suffers from a high complexity of $O(n^2)$

Md Zahidul Islam, Md Anisur Rahman, and Terry Bossomaier are with the School of Computing and Mathematics, Charles Sturt University, Panorama Avenue, Bathurst, NSW 2795, Australia. e-mail: zislam@csu.edu.au, arahman@csu.edu.au, tbossomaier@csu.edu.au. Web: http://csusap.csu.edu.au/~zislam.

Vladimir Estivill-Castro is with the School of Information and Computation Technology, Griffith University, Kessels Rd, Nathan, QLD 4111, Australia. e-mail: v.estivill-castro@griffith.edu.au.

Manuscript received February 16, 2017; revised March 16, 2017.

time. Surprisingly, many existing clustering techniques are using genetic algorithms with an even worse complexity of $O(n^3)$ time. These include ACCA (Bhattacharya & De, 2010), HC (Pirim, Eksioglu, Perkins, & Yuceer, 2012) and ACAD (Chowdhury & Das, 2012). Most of these algorithms are randomised; thus, if properly implemented, they ought to be executed several times, and the best result of such repetitions is to be taken as an answer.

Thus, comparing an algorithm A that is $O(n)$ but potentially delivers worse clusters than another algorithm B that is $O(n^2)$ needs careful considerations. The reason is that, modulo some constants the apparently faster but worse quality performer A can be repeated $O(n)$ times from different random starting points for each execution of B with apparent higher quality clustering results but costing $O(n^2)$. Care should be taken when contrasting cluster quality with computational effort between randomised algorithms, and this is a point we emphasise in this paper. We show that with exactly the same computational effort for multi-start versions of all the algorithms discussed, the proposed GENCLUST++ here produces clustering of higher quality than the other methods. We show this across 18 commonly used datasets, and we validate our results with a statistical analysis that establishes the statistical significance of our results.

1.1 Notation and Definitions

In order to present the main contributions of this paper we shall first introduce basic notations and definitions. We represent a dataset by a sequence $R = \langle R_1, R_2, \dots R_{|R|} \rangle$ of $|R| = n$ records, where two records R_i and R_j can be exactly same. Records are members of the Cartesian product $A_1 \times A_2 \times \dots \times A_m$ of $|A| = m$ attributes. We denote the set of attributes by $A = \{A_1, A_2, \dots A_m\}$. An attribute A_j can be either numerical or categorical. If A_j is numerical, then its domain $A_j = [l, u]$ varies between a lower limit l and an upper limit u where the domain size is the number of possible values between l and u. For a numerical attribute (such as "Number of Students") with only integer values, the domain size can be computed as $|u - l + 1|$. If A_j is categorical, then its domain is presented as $A_j = \{a_j^1, a_j^2, \dots a_j^d\}$ where the domain size is $|A_j| = d$. R_i represents the i-th record, $R_{i,j}$ represents the j-th attribute value of the i-th record, and $R_{i,j} \in A_j$.

We now define the terminology pertinent to the genetic algorithm for clustering. By a population P_s we mean a sequence of chromosomes (Liu et al., 2011; Rahman & Islam, 2014) (again, in a population duplicates are possible). A chromosome encodes a clustering solution as a set $C = \{c_1, \dots, c_k\}$ of genes i.e. cluster centers. All records for which gene c_s is the closest among all genes form the cluster C_s (and in that case c_s can be considered as the representative of those records). A chromosome having a set of genes is a complete clustering solution with as many clusters as the number of genes. Different chromosomes may have different number of genes. Each record is assigned to one and only one gene of a chromosome to form the clusters. That is, we only consider crisp clusterings.

Typically, each gene/cluster center is a record, where all attribute values are drawn from the domains of the attributes. For a numerical attribute A_j, the attribute value of the s-th gene is the numerical mean of the data records assigned to the representative gene: $\frac{\sum_{R_i \in C_s} R_{i,j}}{|C_s|}$, where $|C_s|$ is the number of records in cluster C_s. For a categorical attribute A_k, the attribute value of the gene is the mode of A_j of all records belonging to cluster C_s.

1.2 The main contributions

In this paper we propose a genetic algorithm called GENCLUST++ with several novel components to achieve high quality clustering solutions with $O(n)$ time complexity. Unlike many existing techniques (Lloyd, 1982; Arthur & Vassilvitskii, 2007; Rahman & Islam, 2012; Rahman, Islam, & Bossomaier, 2014), it does not require any user input on the number of clusters k or radius of a cluster r. Moreover, it identifies a set of high quality chromosomes as the initial population — instead of using a set of random chromosomes.

GENCLUST++ has only $O(n)$ complexity for the initial population selection, unlike GENCLUST which has $O(n^2)$ complexity for this. GENCLUST uses a series of possible radius values to compute the initial population, where many of these radius values are inappropriate for a data set and thus may not produce sensible cluster/s. Therefore, many chromosomes in the initial population can still be of bad quality. There is also no guarantee that the set of radii used in GENCLUST contains the correct set of radii. Our experimental results suggest that GENCLUST++ outperforms GENCLUST without the need to manage sets of radii for defining a cluster around a center.

To seed in the initial population, GENCLUST++ applies K-MEANS (Lloyd, 1982; Jain, 2010) or K-MEANS++ (Arthur & Vassilvitskii, 2007) multiple times with different k values ranging from $k = 2$ to $k = 10$, since a useful number of clusters k in a dataset generally varies between these values. A quick analysis of the datasets in UCI machine learning repository (Frank & Asuncion, 2010) supports this consideration. Even if this were not the case, human understanding of the data set would prefer such a bounded number of clusters, and perhaps later a dendogram or subdivision could be applied to large clusters. Nevertheless, in order to enable a discovery where the data speak for themselves GENCLUST++ also uses a range of k values between 2 and $\sqrt{|R|}$ (recall $|R| = n$ is the number of records). For each k value, K-MEANS or K-MEANS++ is applied many times in order to explore the data and high fitness chromosomes with different forms and values of k.

GENCLUST++ ranks the k values of the initial solutions, where a k value with good clusters is ranked high. It then chooses the initial population probabilistically with higher probability awarded to chromosomes with higher ranking. Therefore, the chromosomes in the initial population are of high quality, where the exploratory nature of traditional genetic algorithms is still present. In particular, due to various genetic operations such as crossover and mutation over many generations a wide range of different k values are explored, without being limited to only those used in the initial population.

We recently proposed a genetic algorithm for clustering called HeMI (Beg, Islam, & Estivill-Castro, 2016), where we

took a somewhat similar approach for the initial population selection. However, unlike GENCLUST++ HeMI does not pick chromosomes probabilistically; instead HeMI picks only the best chromosomes. Moreover, HeMI picks 50% of the initial population from the pool of chromosomes obtained through K-means using k ranging from 2 to 10, and the remaining 50% from random k. We argue that forcing to pick 50% chromosomes from random k may be problematic. Instead we prefer a probabilistic selection to encourage good chromosomes (whereever they come from) while maintaining some uncertainty to facilitate exploration.

In addition to the new initial population selection, GENCLUST++ presents a couple of novel genetic operators such as the Probabilistic Cloning and short K-MEANS or K-MEANS++, and a new design/arrangement of the genetic operators (see Algorithm 1). GENCLUST++ combines the best features of the fast hill-climbing of K-MEANS and K-MEANS++, with the exploratory nature of genetic algorithms. The idea is that any solution of poor quality can rapidly be polished to a local optima (or near by) using a few iterations of the K-MEANS' hill-climber; however combinations of one or more local-optima can only be achieved by genetic operators. To this end, GENCLUST++ incorporates a regular intervention (every 10 generations) where chromosomes that represent clustering solutions are polished by K-MEANS' hill-climber ensuring that the population has a large number of high performing chromosomes in the population. This benefits the genetic search, since operators like mutation and crossover only improve the fitness stochasticly and with low probability.

To drive the search successfully, we recommend storing the best performing chromosome found so far, and injecting it back to the population. In every 10-th iteration we carry out the probabilistic cloning operation where we probabilistically create a cloned population from a recent population with high quality. It is possible to have multiple cloned (i.e. exactly same) chromosomes in this new population. Therefore, diversity is re-introduced by applying mutation to the cloned chromosomes. The loss of fitness due to the mutation is then repaired by submitting the mutated chromosomes to a short length (15 iterations) of the K-MEANS' hill-climber. We demonstrate experimentally that the cloning and maintaining a population of high performing chromosomes improve the overall quality of the clustering solution.

The elitism of GENCLUST++ is effective. In its first form mentioned above, it re-injects the best performing chromosome after any disruption is applied to a population (or partial population, since we will see that sub-algorithms like CROSSOVER create several intermediate stages of a population) so that a high performer is not lost. However, we include another form of elitism. In the early generations (experimentally we have decided to use the first ten generations), the drastic changes in the population through crossover and mutation in order to explore the solution space are accepted. This enables an initial phase of wide exploration.

However, after the initial phase (10 generations), the new populations is judged against the previous population and chromosomes are not discarded unless there are better ones in the new generation. This still allows a level of exploration (through crossover and mutation) within a generation. However, it reduces the rate of disruptions by genetic operators that result in chromosomes of low fitness impacting on the already high performers. Thus, in later generations when high quality chromosomes dominate the population, the high quality chromosomes are not removed and some search efforts are allowed to be wasted. A final state of polishing is performed to the best chromosomes of the last generation. This ensures that from the perspective of K-MEANS this is an excellent solution. All these novel components make the proposed GENCLUST++ very effective.

In Section 2 we discuss a number of relevant techniques. Section 3 and Section 4 presents our proposed technique and experimental results, respectively. We then present concluding remarks in Section 5.

2. LITERATURE REVIEW

The statistical theory of multivariate analysis of variance is an inductive principle for clustering that aims at minimizing the intra-cluster distances and maximizing the inter-cluster distances. The particular formulation of this is to measure the trace of the total scatter matrix. This is equivalent to minimizing the total squared error, namely

$$\min L_2(C) = \sum_{i=1}^{n} \text{Euclid}^2(R_i, \text{Rep}[R_i, C]). \tag{1}$$

Where $C = \{c_1, \ldots, c_k\}$ are k centers, and the representative $\text{Rep}[R_i, C]$ of R_i in C is the closest c_j to R_i. Except when $k = 1$, where the answer is the mean, optimizing Equation (1) is a computationally hard problem.

K-MEANS (Lloyd, 1982) is an iterative method that applies a simplified version of Expectation-maximization (EM) to heuristically hill-climb and approximate a solution for a given value k. The classical starting point is to randomly chose k points in the data R as the initial set C_0. With a set C_i of centers or seeds, a partition of the data R is computed by finding for each R_i, is representative center: $\text{Rep}[R_i, C]$. Once we have a label for each R_i, we find a new center as the mean of all those R_i that have the same representative (and thus attributes need to be numeric):

$$c_j^{i+1} = \sum_{R_i \mid c_j^i = \text{Rep}[R_i, C]} \frac{R_i}{\|\{R_i \mid c_j^i = \text{Rep}[R_i, C]\}\|}.$$

This is considered to be one iteration of K-MEANS and, with careful programming, both stages (labeling the data and re-computation of the new centers) can be performed with one pass through the data. In can be proven formally that both stages improve the value of the measure $L_2(C)$, formally establishing K-MEANS' hill climbing nature. Thus, iterations continue until at least one of the termination conditions is met. Typically one termination condition is a user defined maximum number of iterations. Another termination condition is the improvement of objective functions in two consecutive iterations is less than a user defined threshold.

It has been known for a long time that the final clustering quality depends heavily on the quality of the initial seeds.

Remarkably, recently it has been shown that one can guarantee an approximation ratio on the objective function: K-MEANS++ (Arthur & Vassilvitskii, 2007) aims to choose good quality initial seeds in order to produce better quality final clustering solutions. It also takes as input the value k and then randomly chooses a record as the first seed. The second seed is chosen probabilistically where the probability of a record being chosen as the 2nd seed is proportionate to its distance from the first seed. That is, the record having the maximum distance with the first seed has the highest probability of being chosen as the 2nd seed. The 3rd seed is again chosen probabilistically where the record having the highest distance with the seed closest to it has the maximum probability. The probabilistic seed selection process continues until k seeds are selected.

While K-MEANS++ aims to reduce the randomness of k-means in the selection of initial seeds and gives a guarantee on the approximation ratio, such approximation ratio is $O(\log n)$, which although theoretically informative, it may still be very much of the mark for large n. K-MEANS++ with its wiser selection of initial starting points reduces diversity, thus it is unclear that repeating K-MEANS++ over and over again results in overall better solutions. Since K-MEANS and K-MEANS++ use the same hill-climber approach, they obtain very rapidly the local optima near the initial set C_0 of centers. Such local-optima have been reported in the literature as usually being of poor quality (Maulik & Bandyopadhyay, 2000; Xiao, Yan, Zhang, & Tang, 2010). The hybridization of genetic algorithm with hill-climbers can improve the quality of the approximation and thus the cluster quality. Incorporating K-MEANS into genetic search typically improves the quality of clusters (Maulik & Bandyopadhyay, 2000; Xiao et al., 2010; Hruschka, Campello, Freitas, & de Carvalho, 2009). Clustering techniques based on genetic algorithms also typically do not require a user input on the number of clusters k. For example, AGCUK (Liu et al., 2011) automatically identifies the number of clusters.

AGCUK selects the number of genes (i.e. seeds) of a chromosome randomly. It also randomly chooses records as genes of a chromosome, for the initial population that has a user defined number of chromosomes (Liu et al., 2011). Due to the random selection of genes, the chromosomes in the initial population may not contain genes representing all clusters of the dataset. The crossover operation does not create new genes instead it only re-arranges the genes of parent chromosomes. The mutation operation slightly changes some genes and thereby in a way creates new genes. However, the mutation operation typically performs such a small change that new genes are still similar to original genes. Therefore, it is important to contain genes representing all clusters, especially when AGCUK does not apply K-MEANS on chromosomes.The absence of all genes in the initial population may cause a drop in the final clustering quality. This has been illustrated in the literature (Rahman & Islam, 2014).

Another drawback of AGCUK is that it does not re-arrange the genes before a crossover operation. The crossover operation takes two parent-chromosomes as input and randomly divides each chromosome into two parts: the left and right part. The right part of the second chromosome joins the left part of the first chromosome to form the first child-chromosome. The remaining two parts join to form the second child-chromosome. If the genes in the parent-chromosomes are sequenced in such a way that the left part of the first parent-chromosome and the right part of the second parent-chromosome represent the same area (i.e. clusters) of the dataset then the genes of the first child-chromosome will represent only one area of the dataset. The child-chromosome will not have genes representing the other area (i.e. clusters) of the dataset. This has also been illustrated in the literature (Rahman & Islam, 2014).

In order to avoid this GAGR (Chang et al., 2009) rearranges the genes before crossover. However, the gene rearrangement technique used in GAGR requires both parent-chromosomes to have the same size i.e. the same number of genes. GENCLUST (Rahman & Islam, 2014) uses a rearrangement approach that can handle chromosomes of different size.

GENCLUST also carefully selects high quality initial population which was experimentally shown to be effective. While clustering techniques based on genetic algorithms (Chang et al., 2009; Liu et al., 2011) typically work on datasets having only numerical attributes, GENCLUST can handle both numerical and categorical attributes.

GENCLUST has the following drawbacks. First, its initial population selection technique has a time complexity of $O(n^2)$ which can be problematic for big datasets. Second, it uses a set of user defined radii of clusters to obtain the initial population. The actual clusters may have radii values that vary from one dataset to another perhaps depending on several factors including the dimension (i.e. number of attributes) of a dataset. The radii of a good clustering may not appear in the user's initial supplied set and therefore the initial population may not have genes representing several existing clusters in the dataset. Third, since GENCLUST does not use any health check operation, several healthy chromosomes of the initial population can quickly be lost (i.e. not preserved) due to the crossover operation. Selection with GENCLUST ignores earlier chromosomes when building a new generation. This is another reason why the healthy chromosomes of the initial population are not preserved.

While preserving chromosomes with high fitness reduces diversity and genetic explorations, the presence and preservation of such healthy chromosomes increases the speed of convergence (Othman, Deris, Illias, Zakaria, & Mohamad, 2006). Many genetic algorithms aim to preserve healthy chromosomes by not applying the mutation operation on good chromosomes (Chang et al., 2009; Rahman & Islam, 2014).

3. GENCLUST++: THE PROPOSED CLUSTERING TECHNIQUE

3.1 Basic Ideas

We apply several new ideas into a vanilla genetic-search to produce a high quality clustering method. The first idea is to start GENCLUST++ with a set of good quality chromosomes, instead of using randomly chosen chromosomes. By using the

hill-climber of K-MEANS we can achieve a set of reasonably good chromosomes. Therefore, GENCLUST++ uses several different k-values and for each k-value it runs several instances of MK-MEANS (Rahman & Islam, 2014) or MK-MEANS++. These are variants of K-MEANS and K-MEANS++ that can handle both numerical and categorical attributes; details will be provided below when we discuss the components of GENCLUST++. This initial exploration provides several values of k that result in good clusterings. It then seeds its initial population of clustering solutions with the best k-values.

The second type of intervention applies elitism to the vanilla genetic algorithm. It consists of improving the average population fitness at regular intervals or epochs by cloning chromosomes (i.e. chromosomes with high fitness values) and replacing low performing chromosomes by the clones of high fitness chromosomes. Cloned chromosomes undergo a mutation operation to introduce divergence among them. The diverse chromosomes then undergo a short length (15 iterations) MK-MEANS. The application of a few iterations of the hill-climber of MK-MEANS is justified because generally the K-MEANS hill-climber reaches a reasonably good clustering solution within a small number of iterations (Rahman et al., 2014). Moreover, the aim of the MK-MEANS here is not to produce a final clustering solutions, but to repair any slight fitness damage caused by the mutation operation to maintain a population of high achievers. We refer to the application of the hill-climber as polishing.

Our algorithm also compares chromosomes between two consecutive generations. Between ancestor chromosomes and off-spring chromosomes, the algorithm picks the best chromosomes from them for the next resulting generation. This ensures that the impact of the elitism is preserved and strong offsprings of cloning and polishing does improve the population fitness and that again, the random crossover and mutation operations do not introduce very low performers.

Our technique allows a user to specify the fitness function. Naturally, the evaluation criteria that a user plans to use in the cluster quality analysis could be used as the fitness function in order to get a good clustering solution. For example, if a user plans to evaluate the clustering solution by *DB* Index (Tan, Steinbach, & Kumar, 2005), then *DB* Index could be used as the fitness function in our algorithm. However, our experimental results show that clustering solutions produced by our technique also achieve high quality clustering results according to metrics other than the one used as a fitness function.

3.2 Main Components

Our proposed algorithm GENCLUST++ has the following eight main components (see Algorithm 1 for the pseudo-code of GENCLUST++ where the components are identified with comments).

Component 1: Normalize the dataset.
Component 2: Modified K-means called MK-means.
Component 3: *Initial Population with Probabilistic Selection.*
Component 4: Crossover.
Component 5: Elitism.
Component 6: Mutation.
Component 7: *Probabilistic Cloning with M*K-MEANS.
Component 8: *Chromosome Selection*

The components in *italic* font are contributions of this paper. Moreover, the overall arrangement of all components in Algorithm 1 to achieve the higher clustering quality is another contribution of the paper. There are other modules that are encapsulated as functions for Algorithm 1. We proceed to describe the components in order.

Algorithm 1: GENCLUST++

Input : A dataset R_o, and a user defined number of generations N
Output: A set of clusters C

```
/* Compo. 1: Normalize the dataset */
R ← Normlize(R_o);
/* Compo. 3: Initial Population with
   Probabilistic Selection          */
P_f ← InitialPopulation(R);
CR_b ← BestChromosome(P_f, F_f = fitness(P_f, R));
P_s ← ProbabilisticSelection(P_f, R);
for g ← 1 to |N| do
    if g mod 10 ≠ 0 then
        /* Compo. 4: Crossover          */
        P_c ← CROSSOVER(P_s);
    else
        /* Compo. 7: Cloning with
           Mutation and MK-MEANS        */
        P_cl ← ProbabilisticCloning(P_s, R);
        /* Compo. 5: Elitism            */
        CR_b ← Elitism(CR_b, P_cl, R);
        P_c ← MK-MEANS(P_cl, R, 15);
    /* Compo. 5: Elitism                */
    CR_b ← Elitism(CR_b, P_c, R);
    /* Compo. 6: Mutation               */
    P_m ← Mutation(P_c, R);
    /* Compo. 5: Elitism                */
    CR_b ← Elitism(CR_b, P_m, R);
    if g > 10 then
        /* Compo. 8: Chromosome
           Selection                    */
        P_s ← SelectChromosome(P_s, P_m, R)
    else
        P_s ← P_m
Component 2 /* Compo. 2: MK-MEANS       */
C_n ← MK-MEANS(CR_b, R, 50);
C ← Denormalize(C_n, R, R_o);
return C
```

3.2.1 ***Component 1****: Normalize the Dataset:* In Algorithm 1, the input is the original sequence of records, as is, and is denoted by R_o. Its attributes can be numerical and/or categorical. The sequence R_o has $|R_o| = n$ records $R_o = \langle R^o_1, R^o_2, \ldots R^o_{|R_o|}\rangle$. $R^o_{i,j}$ denotes the j-th attribute value of the i-th record in R_o.

The numerical attributes of R_o are normalized. For a numerical attribute $A_j = [l, u]$, $R^o_{i,j}$ is normalized as $R_{i,j} = \frac{R^o_{i,j} - l}{u - l}$. In Algorithm 1, the normalized form of R_o is denoted as R where all numerical attributes of R_o are normalized to the

domain $[0, 1]$.

This normalization assumes the influence of each numerical attribute the same, avoiding issues of scale or units of measurement for distance calculations. We will also place a similar range for categorical attributes to account equally for numerical and categorical attributes.

Algorithm 2: Initial Population

Input : A dataset R, population size s of genetic algorithm

Output: A set P_f of high quality chromosomes from which to sample the initial population

```
Set P_f ← φ; K = ⟨2, 3, 4, ... (3 × s/10 + 1)⟩;
for j = 1 to 5 do
    for i = 1 to |K| do
        /* Produce a chromosome CR
           with K_i genes;
           K_1 = 2, K_2 = 3 ... K_9 = 10        */
        CR ← ProduceChromosome(R, K_i);
        P_f ← P_f ∪ CR;
for i = 1 to 3 × s/10 do
    /* produce a random number in
       the range 2 to √|R|                      */
    k = ProduceRandNumber (2, √||R||);
    for j = 1 to 5 do
        CR ← ProduceChromosome(R, k);
        P_f ← P_f ∪ CR;
return P_f
```

3.2.2 ***Component 2****: Modified* K-MEANS *(M*K-MEANS*):* The modified K-MEANS (Rahman & Islam, 2014) can handle both numerical and categorical attributes while clustering the records. The first thing we need is a metric that combines numerical and categorical attributes. In the modified K-MEANS, the distance between two records R_i and R_l is $d(R_i, R_l) = \frac{\sum_{j=1}^{t} |R_{i,j} - R_{l,j}| + \sum_{j=t+1}^{m} d(R_{i,j}, R_{l,j})}{|A|}$, where attributes are reordered so the first t attributes are numerical, and the next $|A| - t = m - t$ attributes are categorical.

Note that the distance among numerical attributes is the Manhattan metric and not the Euclidean metric. The Manhattan metric is the total absolute error. It is less sensitive to outliers that the total squared error.

For categorical attributes $d(R_{i,j}, R_{l.j}) = 1 - S(R_{i,j}, R_{l.j})$, where $S(R_{i,j}, R_{l.j})$ is the similarity between two categorical values $R_{i,j}$ and $R_{l,j}$ belonging to an attribute A_j. Similarity between two categorical values, $S(R_{i,j}, R_{l.j})$ is computed using an existing technique (Giggins & Brankovic, 2012) and ranges between 0 and 1. Hence, the distance $d(R_{i,j}, R_{l.j})$ also ranges between 0 and 1. A value near 0 for $S(a_j^x, a_j^y)$ indicates low similarity between two categorical values $a_j^x \in A_j$ and $a_j^y \in A_j$. Similarly, a value near 1 for $S(a_j^x, a_j^y)$ indicates high similarity between a_j^x and a_j^y. With this metric, one can find for each record who is its representative among a set C of centers.

If we have a set of records that share a center from the previous iterations, the new seed/center of a cluster is computed by taking the average for numerical attributes and the mode for categorical attributes, over all records belonging to the cluster. The choice of the mean as the center among numerical attribute values when the distance here is the Manhattan distance may surprise some readers. However, computing the center by the Manhattan distance requires sorting for each dimension/attribute the data records of the clusters (a slower method than computing the average). The mean is a sensible center of location when we have filtered outliers and all attributes have very close range of values (the effect of normalization).

Once these variations are established, the modified K-MEANS (MK-MEANS) works exactly the same way as K-MEANS (Tan et al., 2005). Since K-MEANS++ (Arthur & Vassilvitskii, 2007) uses the same two stages for hill-climbing its solution, we can apply the corresponding extensions to create MK-MEANS++ in the same way.

3.2.3 ***Component 3****: Initial Population with Probabilistic Selection:* Most genetic algorithms for clustering (Chang et al., 2009; Liu et al., 2011) select the initial population randomly where genes and the number of genes in a chromosome are selected randomly. The fitness of such a randomly selected initial population is generally very poor.

A strong and diverse initial population is likely to improve the final clustering result. Hence, GENCLUST (Rahman & Islam, 2014) constructs an initial population with a technique that explores a set of radii. It has two main drawbacks: first, it has a high complexity of $O(n^2)$ where n is the number of records and second, since many of the radius values are inappropriate for a given dataset most of the chromosomes in an initial population are still of poor quality.

To address these issues, we propose an initial population selection technique that has a $O(n)$ complexity and is likely to produce an initial population with most members of high-quality while maintaining diversity. GENCLUST++ will work with a default population size s of 30. The starting population is a set of chromosomes denoted by P_s. Each chromosome is a clustering in that it encodes a value of k, and also k vectors with dimensions as the set A of attributes in the dataset R.

To arrive to the initial population, a set P_f of $3 \times s$ chromosomes will be built first. These are typically 90 chromosomes built in *two stages*. Algorithm 2 builds the set P_f. It is used in GENCLUST++ as a function as part of the process of building the initial population.

The *first stage* focuses in obtaining good clusterings for very common values for k. Many datasets observed in practice have less than 10 clusters. For example, we inspect many public-domain datasets for classification, where records are labeled with a class value. Typically, the number of different values for the class attribute in a dataset is an indication of the number of clusters. We also did this for datasets of public domain proposed for clustering tasks, for example, we analyze 157 datasets in the UCI machine-learning repository (Frank & Asuncion, 2010) for which the size of the class attribute has been provided. We found that the average number of classes of the 157 datasets is 5.36 with a standard deviation

of 5.49. It indicates that typically the number of values of the class attribute varies between 2 and 10. This supports our settings that the number of clusters of a dataset generally varies between 2 and 10.

Thus, in the first stage nine $(3 \times s/10)$ different k values for MK-MEANS/MK-MEANS++ are explored: stage^1 $= \{K_1^1 = 2, K_2^1 = 3, \ldots K_9^1 = 10\}$. The superscript for the constant K indicates the stage. For each $K_i^1 \in$ stage^1 we generate a clustering solution (from R) having K_i^1 clusters using MK-MEANS/MK-MEANS++ with input $k = K_i^1$. In this stage, we use the full length of MK-MEANS/MK-MEANS++; that is, the instance of hill-climbing converges or a default maximum of 50 iterations is set a as limit (in our experiments this was never required).

Such an execution of MK-MEANS/MK-MEANS++ delivers a clustering with K_i^1 clusters and each such clustering solution forms a chromosome with K_i^1 genes. All these 9 clusterings are added to the set P_f. The process is repeated 5 times to produce altogether $9 \times 5 = 45$ chromosomes in the *first stage*.

In the *second stage*, we generate another set stage^2 $= \{K_1^2, K_2^2, \ldots K_9^2\}$, of cluster numbers with also $3 \times s/10$ different values to be used as input to MK-MEANS/MK-MEANS++. Now the focus is potentially many clusters in a solution. Thus, now each K_i^2 is a random number between 2 and $\sqrt{|R|}$. Note that some K_i^2 can be the same as K_j^2 (where $i \neq j$) and/or K_p^1. Again, MK-MEANS/MK-MEANS++ delivers a clustering solution running full length with a $k = K_i^2$ as the input for number of clusters. The clustering solution is also encoded into a chromosome that is added to P_f. For each of the nine different values, the process is also repeated five times. Therefore, the second stage also delivers 45 chromosomes Hence, the total number of chromosomes in P_f after two stages is $|P_f| = 45 + 45 = 90$. This completes the description of algorithm 2.

We now have generated, five times, clusters for $3 \times s/10$ inputs for MK-MEANS/MK-MEANS++ in stage one and also for $3 \times s/10$ different values of k in stage two. We have altogether $6 \times s/10$ values of k: $K = \{K_1^1, \ldots K_{3\times s/10}^1, K_1^2, \ldots K_{3\times s/10}^2\}$. Each of these was used by MK-MEANS/MK-MEANS++ five times, so there are 5 clusterings for each $k \in K$. Let these 5 clusterings be denoted $C_{K,1}, \ldots, C_{K,5}$ in descending order of fitness. Now, each of the $30 \times s/10$ members of P_f has a fitness value (such as *DB* Index). As the representative fitness of each $k \in K$, we take the average fitness of the five solutions. Let

$$T(k) = \sum_{i=1}^{5} \text{fitness of } i\text{-th solution for } k \quad (2)$$

$$= \sum_{i=1}^{5} \text{fitness of } C_{k,i} \quad (3)$$

These T values will deliver a probabilistic selection based on direct proportion for values. Repeatedly, a value of k is chosen with probability

$$\frac{T(k)}{\sum_{k \in K} T(k)}.$$

When a value of k is chosen, the next available solution in order of descending fitness $C_{k,i}$ is added to P_s (the initial population for GENCLUST++). Initially, all $k \in K$ have their five solutions available and when a k is chosen, their next solution is promoted to P_s but becomes ineligible for subsequent selection.

The process repeats s times and thus we get s chromosomes in P_s for the initial population of our genetic algorithm. If a $k \in K$ is chosen more than five times, then we create a new chromosome by running MK-MEANS/MK-MEANS++ once more with input k for the number of clusters.

3.2.4 ***Component 4****: The Crossover Operation:* Like most genetic algorithms, an important component of our GENCLUST++ algorithm is its crossover. We use an existing crossover operation CROSSOVER (Rahman & Islam, 2014) that is reproduced in Algorithm 3. All chromosomes are first sorted in a list L in descending order of their fitness values. That is, the chromosome P_i having the best fitness is placed at the beginning of L and the chromosome P_j having the worst fitness is placed at the end. The function *Sort*() in Algorithm 3 performs this fitness ranking.

The crossover operation is applied on the chromosomes by finding pairs of parents. One parent is always the chromosome at the front of the list L, and thus it is the chromosome with the highest fitness left in L. When chosen, it is removed from L. The second chromosome of the pair is selected from the new L by the roulette wheel technique (Maulik & Bandyopadhyay, 2000) (also known as fitness proportionate selection) where a chromosome P_i is picked with probability $P(P_i) = \frac{\textit{fitness}(P_i)}{\sum_{j=1}^{|L|} \textit{fitness}(P_j)}$ (where $\textit{fitness}(P_i)$ is the fitness of chromosome P_i). Every time a chromosome P_i is selected it is removed from the list $L \leftarrow L \setminus \{P_i\}$. Therefore, after the selection of a pair of chromosomes, the size of L decreases by two. The *PickTwoParentChromosomes*() function implements this selection of two chromosomes for L and is used in Algorithm 3.

Since chromosomes are lists of centers of clusters, and such centers are our genes, we apply a re-arrangement that lines up the genes of the second chromosome of a pair (i.e. the chromosome having inferior fitness) to the gene arrangement of the first chromosome of the pair, using an existing gene rearrangement approach (Rahman & Islam, 2014). The re-arrangement is designed to handle chromosomes even with unequal lengths. The first chromosome is called the reference chromosomes and does not change (Rahman & Islam, 2014) while the second chromosome is called the target chromosome. The gene re-arrangement operation's general work-flow is as follows (but details appear in the original presentation of GENCLUST (Rahman & Islam, 2014, Algorithm 1)).

Among all genes of the target chromosome, the gene having the minimum distance with the first gene (i.e. the gene in the 1st spot) of the reference chromosome is placed at the front (i.e. in the 1st spot) of the rearranged version of the target chromosome. This gene is then removed from the target chromosome. From the remaining genes of the target chromosome, the gene having the minimum distance with the gene in the 2nd spot of the reference chromosome is then

placed at the 2nd spot of the rearranged version of the target chromosome. This gene is also then removed from the target chromosome. This process continues until we run out of genes in the target chromosome; if the length (i.e. number of genes) of the target chromosome is either equal to or less than the length of the reference chromosome.

If the target chromosome has more genes than the reference chromosome, then the gene re-arrangement is carried out in two phases. In the first phase, x genes of the target chromosome are re-arranged as before; where x is the length of the reference chromosome. In the second phase, the remaining $y-x$ genes of the target chromosome are inserted one by one in the re-arranged version of the target chromosome in such a way so that a gene is inserted next to its closest gene in the re-arranged version of the target chromosome. All other genes of the re-arranged version of the target chromosome are bodily shifted (Rahman & Islam, 2014).

Once the second chromosome of a pair is rearranged, the pair of chromosomes participate in essentially a conventional single point crossover operation. Each parent chromosome is randomly and independently divided into two parts by imposing a partition line in between two genes, where in each part there are one or more genes. The left part of one chromosome joins the right part of the other chromosome to form an offspring chromosome. Similarly, the remaining two parts join together to form the second offspring chromosome. The production of the two offsprings is the function *ConventionalCrossover* in Algorithm 3.

From the crossover of a pair of parent chromosomes we get a pair of offspring chromosomes. Offsprings are added to an initially empty population P_c which is a partial result of crossover. The selection of chromosome pair as parents, the gene re-arrangement of the second parent and the crossover operation is repeated while the current list L is not exhausted.

To obtain a final result from the crossover, we apply a duplicate removal (i.e. twin removal) operation (Rahman & Islam, 2014) to each offspring chromosome in the set P_c. For this, a function to declare two centers (two genes) duplicates is required. Two genes g_i and g_j were considered duplicates if the distance $dist(g_i, g_j) \leq \delta$. In the empirical evaluation of our approach, the value of δ was set to zero (i.e the numerical precision). However, a very small value different than zero could also be supplied as δ for the duplicate removal operation. If a chromosome has only two genes and they are deemed duplicates, then the duplicate removal operation randomly mutates one gene until the genes are no longer considered duplicates. The *DuplicateRemoval* function is where this process is applied.

3.2.5 ***Component 5****: The Elitism Operation:* The main purpose of the *Elitism* operation (Rahman & Islam, 2014) is to re-insert the highest fitness chromosome if an iteration is producing several under-performers. Thus, the best chromosome found so far does not get lost and the final clustering solution can only improve from (or remain the same as) the best chromosome. The best chromosome found so far up until the current generation is stored as CR_b. It is then compared with the worst chromosome of the offspring produced by CROSSOVER (and also after Mutation). If CR_b has better fitness than the worst chromosome of the next generation produced by CROSSOVER, then such worst chromosome is replaced by CR_b. Always, the best chromosome of the next generation is compared with CR_b; if the best chromosome of the next generation is better than current CR_b, then it is stored in CR_b. The *Elitism*() operation is carried out after every CROSSOVER and also after every Mutation (see Algorithm 1).

Algorithm 3: CROSSOVER

Input : A set of chromosomes L'
Output: A set of offspring P_c

Set $P_c \leftarrow \phi$;
$L \leftarrow$ *Sort*(L');
while $|L| > 0$ **do**
  /* select a pair of parent chromosomes P= {P_x, P_y} from L */
  $P \leftarrow$ *PickTwoParentChromosomes*(L);
  $L \leftarrow L \setminus P$;
  /* rearrange a chromosome of P */
  $P \leftarrow$ *RearrangeOperation*(P);
  $O \leftarrow$ *ConventionalCrossover*(P);
  $P_c \leftarrow P_c \cup O$;
$P_c \leftarrow$ *DuplicateRemoval*(P_c) ;
return P_c

3.2.6 ***Component 6****: The Mutation Operation:* The mutation operation randomly changes the genes of a chromosome in order to explore unconventional solutions (Rahman & Islam, 2014). The mutation probability M_i of the i-th chromosome CR_i in P_c is calculated as follows.

$$M_i = \begin{cases} \frac{f_{max} - \textit{fitness}(CR_i)}{2(f_{max} - \overline{f})} & \text{when } \textit{fitness}(CR_i) > \overline{f} \\ 1/2 & \text{when } \textit{fitness}(CR_i) \leq \overline{f} \end{cases}$$

where $\textit{fitness}(CR_i)$ is the fitness of the i-th chromosome in P_c, $f_{max} = \max_{CR_j \in P_c} \textit{fitness}(CR_j)$ is the fitness of the best chromosome in the current population P_c, and $\bar{f} = \sum_{CR_j \in P_c} \textit{fitness}(CR_j)/\|P_c\|$ is the average fitness of the chromosomes in P_c. Note that f_{max} may not be the same as $\textit{fitness}(CR_b)$.

If a chromosome is chosen for mutation, then we randomly select and (randomly) change an attribute of each gene of the chosen chromosome. The attribute is selected uniformly among the d attributes and the value is selected also uniformly in the corresponding domain.

At the end of the mutation operation the duplicate removal operation is also performed. In this study, duplicate removal is considered to be a part of our mutation operation.

3.2.7 ***Component 7****: Probabilistic Cloning with MK-MEANS:* At every 10th iteration (i.e. the 10th, 20th, 30th etc.) we perform a probabilistic cloning of the chromosomes. We clone chromosomes with probability proportional to the fitness of the chromosomes as per the wheel technique. That is, using fitness proportionate selection, from a population P_s, we generate another population P_{cl} of the same size where the

probability of a chromosome $CR_i \in P_s$ to be inserted into an initially empty P_{cl} is $\frac{\textit{fitness}(CR_i)}{\sum_{j=1}^{|P_s|}\textit{fitness}(CR_j)}$, where $\textit{fitness}(CR_i)$ is the fitness of CR_i. Because the selection is with replacement, a chromosome in P_s may be cloned more than once into P_{cl}, and this is why the operation is called cloning.

At this stage, we apply mutation on all chromosomes in P_{cl} in order to introduce diversity. The cloning and the associated mutation are represented by the $\textit{ProbabilisticCloning}(P_s)$ function in Algorithm 1.

The cloning accelerates convergence, and thus it is not applied in every generation. We found experimentally that a suitable cloning intervention is every 10 iterations. The application of the other genetic operators, specially crossover, often results in chromosomes with lower fitness in the population. These under-performers enable exploration of the search space, but if after 10 iterations the corresponding niche has not been found, poor performers will be probabilisticly removed by the cloning of high fitness chromosomes. Our cloning operation is expected to remove low fitness chromosomes and replace them with chromosomes with high fitness.

Elitism is performed every time we intervene with cloning (see Algorithm 1) We follow this with 15 iterations of MK-MEANS on each chromosome of the population driving the fitness high by hill-climbers (exploitation). The cloning and its mutation produce chromosomes that are required to reach levels of high clustering quality, and this is achieved by the MK-MEANS iterations because the MK-MEANS operation repairs the potential damage of mutation while sustaining the diversity also produced by mutation.

We choose 15 iterations (short length) of MK-MEANS since this is a value regarded in the literature (Rahman et al., 2014) as the number of iterations where K-MEANS makes the steepest improvement to reach a reasonable solution. Since the chromosomes produced by this component will be modified by the next steps (such as crossover and mutation) of the algorithm anyway, we do not require to specialize them to the extreme by running K-MEANS for longer iterations.

As always, elitism is again performed (see Algorithm 1) so this cloning process gets an opportunity to contribute to the best chromosome CR_b found so far. Finally, also within this component, one more pass of the mutation operation is applied to all chromosomes, in order to increase diversity as without it, this step generally reduces such diversity. And once more, elitism is applied immediately after to ensure the best chromosome CR_b found so far is never lost.

3.2.8 ***Component 8*** *: Chromosome Selection:* The crossover operation drastically modifies the chromosomes due to its randomness. Crossover generally replaces all chromosomes of a previous generation even if some of them were of high quality. We allow this to happen in the first nine generations in order to facilitate an exploration of the solution space. In the tenth generation we perform the probabilistic cloning operation. Hence in the 10th generation, the crossover operation does not happen. But, from this generation onward we also carry out the Chromosome Selection operation (Component 8) which merges all chromosomes between two populations of size s: the last (most recent) population and the resulting generation from all operations (see Algorithm 1). This second type of elitism chooses the highest fitted s chromosomes from the merged populations. This restricts radical disruption by the genetic operators, but it still enables their exploratory nature to drive the genetic search.

3.3 Main Steps of GENCLUST++

We now integrate the earlier components and briefly describe the pseudo-code in Algorithm 1 for GENCLUST++ . The inputs are the dataset R_o and the number N of generations (the default is $N = 60$). We denote by P_f the large sequence of chromosomes that Component 3: Initial Population produces before it narrows it down to the running population P_s. A population delivered after crossover (nine out of ten times) is P_c, while once out of ten generations the cloning, mutation and short K-MEANS delivers P_{cl}. The mutation result is P_m, and from the 10th generation onwards we apply chromosome selection delivering P_s. On the next iteration this P_s becomes the new P_s.

At all times the chromosome having the best fitness among those explored is remembered as CR_b. Finally, after the genetic generations reach their limit, the best chromosome is used as the initial solution of a final full-length MK-MEANS to deliver the final clustering solution.

4. EXPERIMENTAL RESULTS AND DISCUSSION

4.1 Datasets

We use 18 publicly available datasets (Frank & Asuncion, 2010) in our experiments. Table I presents some basic information on the datasets. For example, the Glass Identification (GI) and Dermatology (DT) datasets (see Row 1 and Row 2 of the table) have 214 and 366 records, respectively. Some records may have one or more missing attribute value/s, as indicated in the heading of Column 2. In our experiments, prior to applying any clustering technique on a dataset, we delete all record having a missing value/s. For example, DT has 8 records with missing value/s and therefore, DT has 358 records without missing values (see Column 3). DT has 34 numerical and zero categorical attributes. It has a class attribute with domain size 6. Note that class attributes are removed from the datasets before we apply any clustering technique on the datasets since in real life a clustering technique is applied on datasets without any class labels.

Table I shows that there are 12 datasets having only numerical attributes, four (4) datasets having only categorical attributes and two (2) datasets having both numerical and categorical attributes. The MGT dataset has 19,020 records, the PBRHD dataset has 10,992 records and the MR dataset has 8,124 records. There are eight (8) datasets with more than 1,000 records.

4.2 Experimental Setup

For all experiments here the parameters used for GENCLUST++ are a maximum total of 60 generations, generation

TABLE I
A BRIEF INTRODUCTION TO THE DATASETS

Name of dataset	Total No. of records	No. of records without missing values	No. of numerical attributes	No. of categorical attributes	Class size
Glass Identification (GI)	214	214	10	0	7
Dermatology (DT)	366	358	34	0	6
Liver Disorder (LD)	345	345	6	0	2
Mammographic Mass (MM)	961	830	5	0	2
Page Blocks Classification (PBC)	5,472	5,472	10	0	5
Pima Indian Diabetes (PID)	768	768	8	0	2
Statlog Vehicle Silhouettes (SVS)	846	846	18	0	4
User Knowledge Modelling (UKM)	403	403	5	0	4
Wine Quality (WQ)	4,898	4,898	11	0	7
Yeast	1,484	1,484	8	0	10
Pen-Based Recognition of Handwritten Digits (PBRHD)	10,992	10,992	16	0	10
MAGIC Gamma Telescope (MGT)	19,020	19,020	10	0	2
Chess King-Rook vs. King-Pawn (CKRKP)	3,196	3,196	0	36	2
Contraceptive Method Choice (CMC)	1,473	1,473	2	7	3
Credit Approval (CA)	345	345	6	9	2
Heberman (HM)	306	306	0	3	3
Mushroom (MR)	8,124	5,644	0	22	2
Tic-Tac-Toe (TTT)	958	958	0	9	2

size $|P_s| = 30$, thus $|P_f| = 90$. The number of short K-MEANS iterations after cloning is 15, while the final K-MEANS on the best chromosome CR_b has its number of iterations bound to 50.

For all other existing algorithms we use the values as recommended in the original publications. Therefore, The number of chromosomes in the population of GENCLUST (Rahman & Islam, 2014) is set to 30. We run GENCLUST over the recommended limit of 60 generations and maximum 50 iterations of MK-MEANS.

For all experiments and all algorithms that use some form of hill-climber of K-MEANS, we impose the termination condition that the difference of SSE of two consecutive iterations is less than 0.005. The fitness function for both GENCLUST++ and GENCLUST is *DB* Index (Tan et al., 2005).

In AGCUK (Liu et al., 2011), the number of chromosomes in a population and the number of generations are 20 and 50, respectively as recommended in the original study. The values of r_{max} and r_{min} in AGCUK are 1 and 0, respectively. In GAGR (Chang et al., 2009), the number of chromosomes in the initial population is 50 and the number of generations is 50. The fitness functions used in AGCUK and GAGR are *DB* Index and SSE.

4.3 Experimental Results

Since the clustering solutions obtained by the techniques can vary between different runs, we run each technique ten (10) times on each dataset recording their cluster quality. The tables present the average results of the ten (10) clustering solutions on each dataset for each technique. In contrast to the tables, we use figures to present the average results over all datasets and over all repetitions for a dataset. The discussion of the figures will be followed by a discussion of the statistical significance analysis we performed through the non-parametric sign test analysis (Triola, 2001).

We compare the performance of our proposed GENCLUST++ with nine published alternatives: GENCLUST-H (Rahman & Islam, 2014), GENCLUST-F (Rahman & Islam, 2014), AGCUK (Liu et al., 2011), GAGR (Chang et al., 2009), SABC (Ahmed & Dey, 2007), GFCM (Lee & Pedrycz, 2009), KL-FCM-GM (Chatzis, 2011), K-MEANS++ (Arthur & Vassilvitskii, 2007) and K-MEANS (Lloyd, 1982). Some of these techniques can only handle numerical attributes. So, we use the first 12 datasets (see Table I) for the comparisons between GENCLUST++ and those methods that use only numerical attributes. Such comparison on datasets with only numerical values first uses *DB* index (see Table II) since the *DB* Index is used as the fitness function in a several of the techniques being compared: GENCLUST++, GENCLUST-H, GENCLUST-F and AGCUK. The best average *DB* Index (among all techniques) for a dataset is shown in **bold** font.

We can see from Table II that GENCLUST++ achieves the best average *DB* Index among all techniques in 9 out of 12 datasets. On an average over all twelve datasets, Figure 1 shows that GENCLUST++ achieves the best average *DB* Index. Although K-MEANS achieves the best average *DB* Index among all techniques in 2 out of 12 datasets, it has a very high average *DB* Index over all datasets (see again Figure 1).

Although the fitness function used in GENCLUST++ is the *DB* Index we next compare its effectiveness with the existing techniques in terms of two other fitness functions: *COSEC* (Rahman & Islam, 2014) and *XB* Index (Tan et al., 2005). The main goal here is to examine whether or not the clusters obtained by GENCLUST++ are better than those obtained by other techniques in terms of a fitness function that is different from the fitness function used in GENCLUST++. Moreover, since none of the techniques (in our experiments) uses *COSEC* and *XB* Index as the fitness function, this is a neutral evaluation of techniques being reviewed. Figure 2 shows that our proposed GENCLUST++ achieves the best

TABLE II
DB INDEX(LOWER THE BETTER) OF THE TECHNIQUES ON THE 12 NUMERICAL DATASETS

Datasets	GenClust++	GenClust-H	GenClust-F	AGCUK	GAGR	K-MEANS
DT	**1.0946**	1.1759	1.4950	1.2307	2.1240	2.1687
GI	**1.2452**	1.5079	1.4221	1.4563	1.3367	1.3963
LD	**1.0978**	1.1295	1.4352	1.2080	1.7698	1.6065
MM	**0.5970**	0.6067	0.6076	0.7918	1.4228	1.3025
PBC	**0.9131**	0.9699	1.3005	1.0220	1.7854	1.1902
PID	1.9693	1.9788	2.0549	2.4310	1.8670	**1.8120**
SVS	**0.8911**	0.9001	0.9867	0.9074	1.5522	1.5392
UKM	2.1653	2.2752	2.2631	2.1594	1.8721	**1.7914**
WQ	**0.8652**	1.8070	2.2953	1.4915	1.5506	1.8581
Yeast	**1.5079**	1.5944	1.8663	1.6315	1.9571	1.7531
PBRHD	1.5394	**1.3724**	3.2410	1.5685	1.8295	1.4384
MGT	**1.2072**	1.2582	1.3624	1.2630	2.2598	1.8810

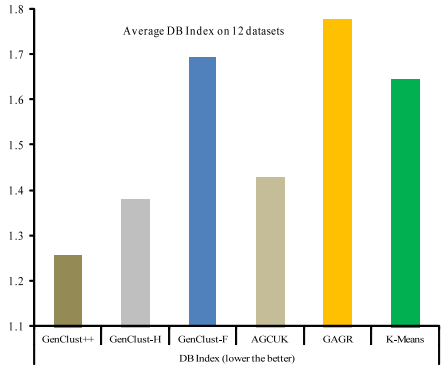

Fig. 1. Average *DB* Index of the techniques for 12 datasets, the smaller value the better the clustering result.

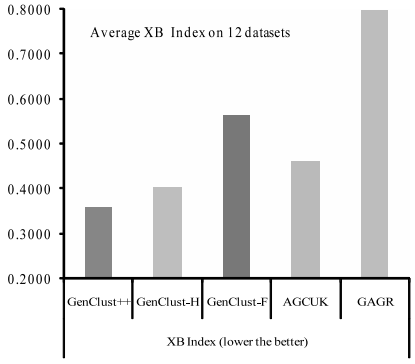

Fig. 3. Average *XB* Index of the techniques for 12 datasets, the smaller the better.

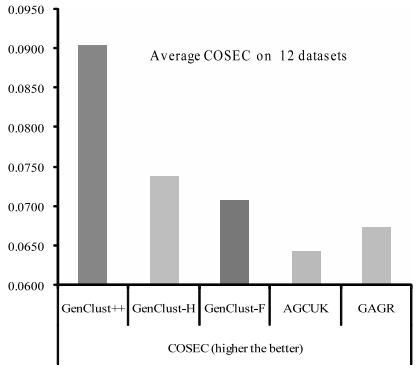

Fig. 2. Average *COSEC* of the techniques for 12 datasets, the larger the better.

average *COSEC* value among all techniques over 12 datasets. Figure 3 shows that GENCLUST++ also achieves the best average *XB* Index among all techniques. Note that for every dataset, we run every technique 10 times and the figures show the average of all datasets and runs.

Now, we present experimental result on all 18 datasets including those having categorical attributes. We exclude GAGR and AGCUK as they require datasets having no categorical attributes and compare GENCLUST++ with GENCLUST-H and GENCLUST-F (Rahman & Islam, 2014) (see Table III). In 17 out of 18 datasets GENCLUST++ achieves the best average *DB* Index results — from 10 runs on each dataset.

TABLE III
DB INDEX (LOWER THE BETTER) OF THE TECHNIQUES ON THE 18 DATASETS.

Datasets	GENCLUST++	GENCLUST-H	GENCLUST-F
DT	**1.0946**	1.1759	1.4950
GI	**1.2452**	1.5079	1.4221
LD	**1.0978**	1.1295	1.4352
MM	**0.5970**	0.6067	0.6076
PBC	**0.9131**	0.9699	1.3005
PID	**1.9693**	1.9788	2.0549
SVS	**0.8911**	0.9001	0.9867
UKM	**2.1653**	2.2752	2.2631
WQ	**0.8652**	1.8070	2.2953
Yeast	**1.5079**	1.5944	1.8663
PBRHD	1.5394	**1.3724**	3.2410
MGT	**1.2072**	1.2582	1.3624
CKRKP	**2.2968**	2.4741	2.5821
CMC	**1.0246**	**1.0246**	1.0262
CA	**0.2751**	1.2177	1.8548
HM	**1.5609**	1.6587	2.0809
MR	**0.3903**	2.2616	0.6967
TTT	**0.6650**	0.7220	0.7318

Figure 4 shows the overall average of *DB* Index values over all 18 datasets. Similarly, we also present the overall average for *COSEC* and *XB* Index values over all 18 datasets in Figure 5 and Figure 6, respectively. The figures show that the proposed GENCLUST++ method clearly outperforms the existing techniques.

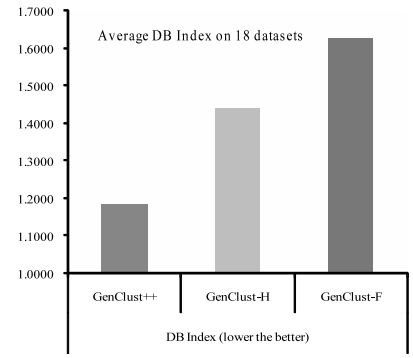

Fig. 4. Average *DB* Index of the techniques for 18 datasets

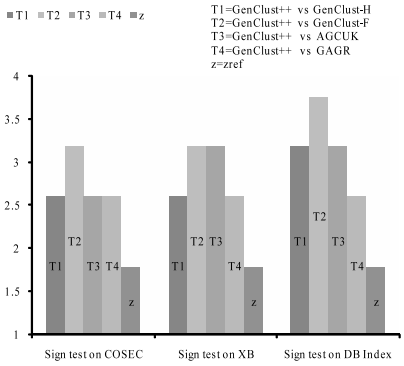

Fig. 7. Sign test of GENCLUST++ on the first 12 datasets.

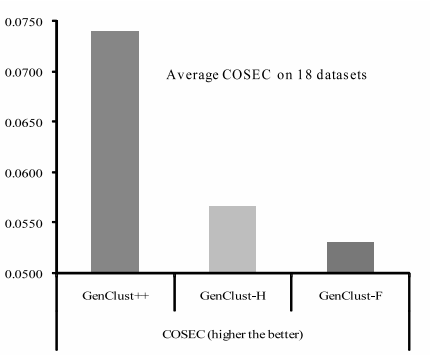

Fig. 5. Average *COSEC* of the techniques for 18 datasets

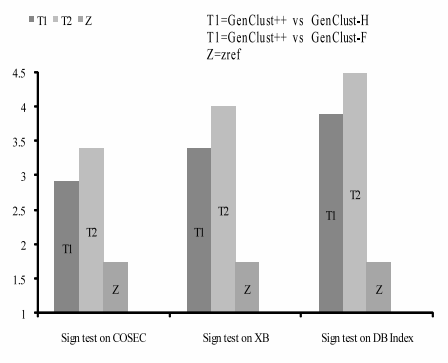

Fig. 8. Sign test of GENCLUST++ on all the 18 datasets

4.4 Statistical Significance Analysis

The results presented so far are strong evidence that our proposed GENCLUST++ outperforms other techniques in the large majority of the datasets. Nevertheless, we now carry out a non-parametric sign test (Triola, 2001) to evaluate the statistical significance of the improvements achieved by GENCLUST++ over the published techniques. The right tailed sign test is carried out for the significance level $\alpha = 0.05$ meaning 95% significance level. Figure 7 presents the sign test results of GENCLUST++ on 12 numerical datasets in terms of three evaluation metrics/criteria: *COSEC*, *XB* Index and *DB* Index. The first four bars (for each evaluation metric) present the $z-$values (test statistic values) for GENCLUST++ performance against the performance of each of four (4) published techniques. The fifth bar presents the z-ref value. If a $z-$value is greater than the z-ref value, then it indicates that the observed superior performances of GENCLUST++ against the corresponding published existing technique is statistically significant. These shows that the observed results of a better *COSEC*, *XB* Index and *DB* Index for GENCLUST++ against each of the other 4 published techniques is a statistical significant finding across the board. In summary, Figure 7 shows that, on numerical data sets, GENCLUST++ achieves significantly better results than all four existing techniques in terms of all three evaluation metric/criteria. Figure 8 shows that the results obtained by GENCLUST++ are significantly better than the results obtained by GENCLUST-H and GENCLUST-F over all 18 datasets and all 3 evaluation criteria.

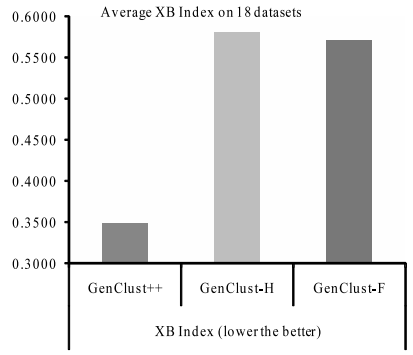

Fig. 6. Average *XB* Index of the techniques for 18 datasets

4.5 Empirical Analysis of Various Components of GENCLUST++

We now evaluate the effectiveness of various proposed components of GENCLUST++, by comparing the results of GENCLUST++ with the results obtained by GENCLUST++ without a particular component. For example, at first we examine the effectiveness of the Component 3; the Initial Population To do so, we replace Component 3 in GENCLUST++ with a simpler component for the initial population selection. The replacement first generates 90 chromosomes. The number of

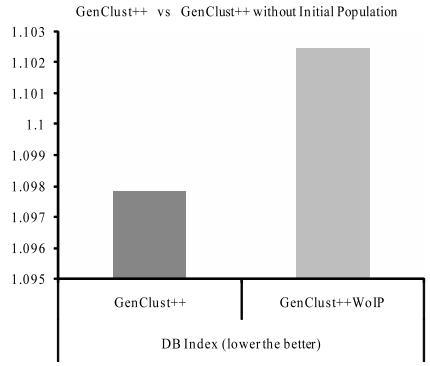

Fig. 9. *DB* Index of GENCLUST++ vs GENCLUST++ without Initial Population on LD dataset

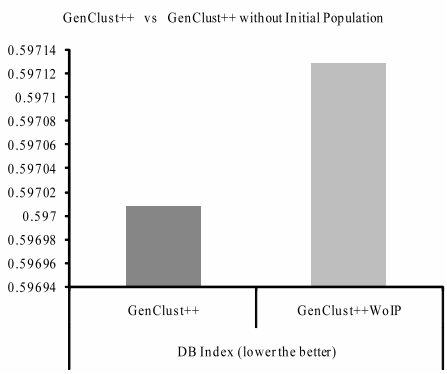

Fig. 10. *DB* Index of GENCLUST++ vs GENCLUST++ without Initial Population on MM dataset

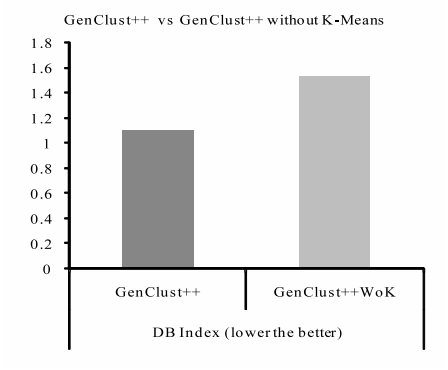

Fig. 11. *DB* Index of GENCLUST++ vs GENCLUST++ without MK-MEANS on LD dataset

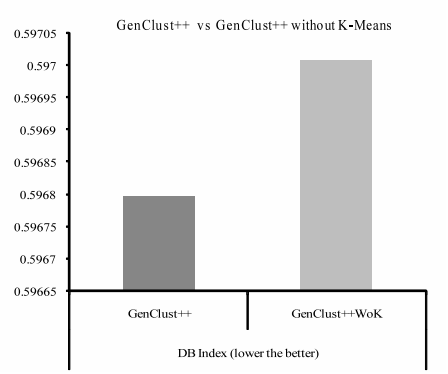

Fig. 12. *DB* Index of GENCLUST++ vs GENCLUST++ without MK-MEANS on MM dataset

genes in each chromosome is randomly selected between 2 and $\sqrt{|R|}$, where $|R|$ is the number of records in a dataset R. A gene in a chromosome is a randomly chosen record, from the dataset R. It then applies a probabilistic approach same as the one used in GENCLUST++ in order to choose 30 chromosomes out of these 90 chromosomes for the final initial population.

All other steps/components remain exactly same as the components of GENCLUST++. The technique with the pseudo component is called "GENCLUST++ without Initial Population (GENCLUST++ WoIP)". Figure 9 and Figure 10 show the average *DB* Index of 10 runs of GENCLUST++ and GENCLUST++ WoIP on the LD and MM datasets. Clearly GENCLUST++ achieves better results than GENCLUST++ WoIP, indicating the usefulness of the initial population component.

We now examine the usefulness of the MK-MEANS at the end of 60 generations. So we slot out the MK-MEANS component from GENCLUST++ and call it "GENCLUST++ without MK-MEANS (GENCLUST++ WoK)". Figure 11 and Figure 12 present the average *DB* Index of 10 runs on the LD and MM datasets, respectively. Clearly, GENCLUST++ performs better *DB* Index than GENCLUST++ WoK on both datasets, indicating the usefulness of MK-MEANS.

Another important component of GENCLUST++ is the probabilistic cloning with MK-MEANS where we aim to improve the overall health of an entire population so that we can get effective results from the crossover, mutation and other operations. We simply remove the probabilistic cloning component; such variant is named "GENCLUST++ WoPC". Figure 13 and Figure 14 indicate the usefulness of the probabilistic cloning component in achieving better clustering results as evidenced from the average *DB* Index of 10 runs on two datasets.

Figure 15 shows the average fitness of the best chromosomes over 18 datasets for every iteration of GENCLUST++ and GENCLUST-H. We can see that both techniques progress similarly up to the 10th iteration. From the 10th iteration GEN-

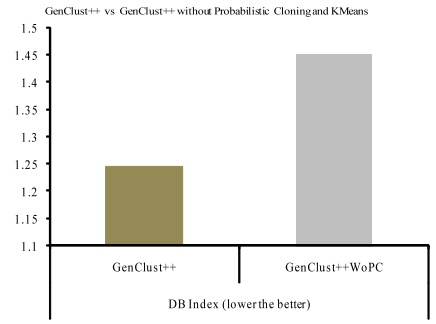

Fig. 13. *DB* Index of GENCLUST++ vs GENCLUST++ without Probabilistic Cloning and K-MEANS on the GI dataset

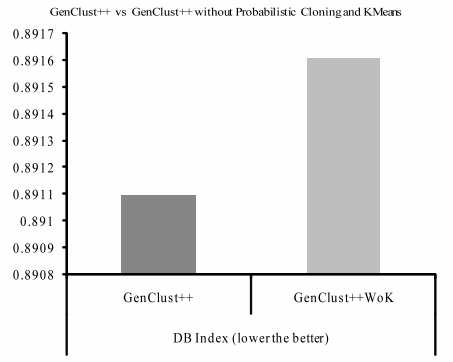

Fig. 14. *DB* Index of GENCLUST++ vs GENCLUST++ without Probabilistic Cloning and K-MEANS on SVS dataset

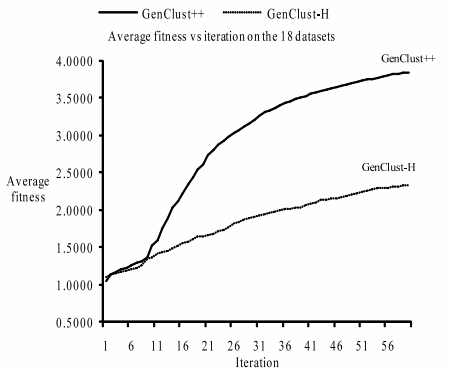

Fig. 15. Average fitness vs iteration of GENCLUST++ and GENCLUST-H over the 18 datasets

CLUST++ makes massive improvement compared to GEN-CLUST. This is also very indicative of the usefulness of the probabilistic cloning and MK-MEANS in the 10th iteration.

5. CONCLUSION

This paper has provided a very effective blend of the hill-climbing optimization of K-MEANS with the exploratory nature of genetic algorithms. The evidence of the experimental evaluation shows that GENCLUST++ provides solutions of higher quality with equivalent computational resources. Moreover, our analysis of significance shows that these results are statisticly meaningful and that the components that constitute GENCLUST++ contribute to the observed performance in obtaining clusterings of higher quality.

Nevertheless, we see some potential for improvement. The introduction of an elitism from generation 10 that ranks the chromosomes of the previous generation against the new generation just to take the top performers is potentially a very abrupt change from an exploratory phase to a exploitation phase. The change is visible in Figure 15 as the separation point between the two curves. Future work would investigate smoothening this change, in a fashion similar to the temperature parameters in simulated annealing. That is, we introduce the contrast between new population and previous population from the first generation, but with a high preference to the new population. With each generation, such preference diminishes until we reach the situation where chromosomes of the new generation must compete in equal circumstances (by rank of fitness value) with chromosomes of the previous generation. We believe this could potentially see a new curve in Figure 15 taking off much earlier, and potentially reaching even higher at a minimal overhead.

ACKNOWLEDGMENT

The authors would like to thank the support provided by their host institutions and in particular the Compact Fund support of the Faculty of Business at Charles Sturt University.

REFERENCES

Ahmed, A., & Dey, L. (2007). A k-mean clustering algorithm for mixed numeric and categorical data. *Data and Knowledge Engineering*, *63*(2), 503 - 527.

Arthur, D., & Vassilvitskii, S. (2007). k-means++: The advantage of careful seeding. In *the eighteenth annual acm-siam symposium on discrete algorithms* (p. 1027 - 1035).

Bandyopadhyay, S., & Maulik, U. (2002). An evolutionary technique based on k-means algorithm for optimal clustering in rn. *Information Sciences*, *146*, 221 – 237.

Beg, A. H., Islam, M. Z., & Estivill-Castro, V. (2016). Genetic algorithm with healthy population and multiple streams sharing information for clustering. *Knowledge-Based Systems*, *114*, 61-78.

Bhattacharya, A., & De, R. K. (2010). Average correlation clustering algorithm (acca) for grouping of co-regulated genes with similar pattern of variation in their expression values. *Journal of Biomedical Informatics*, *43*(4), 560-568.

Brameier, M., & Wiuf, C. (2007). Co-clustering and visualization of gene expression data and gene ontology terms for saccharomyces cerevisiae using self-organizing maps. *Journal of Biomedical Informatics*, *40*(2), 160-173.

Chan, K. Y., Kwong, C. K., & Hu, B. Q. (2012). Market segmentation and ideal point identification for new product design using fuzzy data compression and fuzzy clustering methods. *Applied Soft Computing*, *12*(4), 1371-1378.

Chang, D., Zhang, X. D., & Zheng, C. W. (2009). A genetic algorithm with gene rearrangement for k-means clustering. *Pattern Recognition*, *42*, 1210–1222.

Chatzis, S. P. (2011). The fuzzy c-means-type algorithm for clustering of data with mixed numeric and categorical attributes employing a probabilistic dissimilarity functional. *Expert Systems with Applications*, *38*, 8684–8689.

Chowdhury, A., & Das, S. (2012). Automatic shape independent clustering inspired by ant dynamics. *Swarm and Evolutionary Computation*, *3*, 33-45.

Estivill-Castro, V. (2002, June). Why so many clustering algorithms: A position paper. *SIGKDD Explor. Newsl.*, *4*(1), 65–75.

Frank, A., & Asuncion, A. (2010). *UCI machine learning repository [online available:*

http://archive.ics.uci.edu/ml]. Retrieved from http://archive.ics.uci.edu/ml (Accessed July 7, 2013)

Giggins, H. P., & Brankovic, L. (2012). Vicus - a noise addition technique for categorical data. In *Australasian data mining conference (ausdm 12)* (Vol. 134, pp. 139–148). Sydney, Australia: ACS.

Hruschka, E. R., Campello, R. J. G. B., Freitas, A. A., & de Carvalho, A. C. P. L. F. (2009). A survey of evolutionary algorithms for clustering. *IEEE Transactions on Systems, Man, and Cybernatics*, *39*(2), 133-155.

Jain, A. K. (2010). Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, *31*, 651-666.

Laszlo, M., & Mukherjee, S. (2007). A genetic algorithm that exchanges neighboring centers for k-means clustering. *Pattern Recognition Letters*, *28*, 2359 – 2366.

Lee, M., & Pedrycz, W. (2009, December). The fuzzy c-means algorithm with fuzzy p-mode prototypes for clustering objects having mised features. *Fuzzy Sets and Systems*, *160*(24), 3590 - 3600.

Liu, Y., Wu, X., & Shen, Y. (2011). Automatic clustering using genetic algorithms. *Applied Mathematics and Computation*, *218*(4), 1267–1279.

Lloyd, S. P. (1982). Least squares quantization in PCM. *IEEE Transactions on Information Theory*, *28*(2), 129 –136.

Loai, L., Lin, T., & Li, B. (2008). Mri brain image segmentation and bias field correction based on fast spatially constrained kernel clustering approach. *Pattern Recognition Letters*, *29*(10), 1580-1588.

Maulik, U., & Bandyopadhyay, S. (2000). Genetic algorithm-based clustering technique,. *Pattern Recognition*, *33*, 1455–1465.

Nistér, D., & Stewénius, H. (2006, 17th-22nd June). Scalable recognition with a vocabulary tree. In *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR)* (pp. 2161–2168). New York, NY, USA: IEEE Computer Society.

Othman, R. M., Deris, S., Illias, R. M., Zakaria, Z., & Mohamad, S. M. (2006). Automatic clustering of gene ontology by genetic algorithm. *International Journal of Information Technology*, *3*(1), 37-46.

Pirim, H., Eksioglu, B., Perkins, A. D., & Yuceer, C. (2012). Clustering of high throughput gene expression data. *Computers and Operations Research*, *39*(12), 3046-3061.

Rahman, M. A., & Islam, M. Z. (2011, December). Seed-detective: A novel clustering technique using high quality seed for k-means on categorical and numerical attributes. In *9th australasian data mining conference (ausdm 11)* (Vol. 121, p. 211-220). Ballarat, Australia: CRPIT.

Rahman, M. A., & Islam, M. Z. (2012, December). Crudaw: A novel fuzzy technique for clustering records following user defined attribute weights. In *10th australasian data mining conference (ausdm 12)* (Vol. 134, p. 27-42). Sydney, Australia: CRPIT.

Rahman, M. A., & Islam, M. Z. (2014). A hybrid clustering technique combining a novel genetic algorithm with k-means. *Knowledge-Based Systems*, *71*, 21 - 28. Retrieved from http://dx.doi.org/10.1016/j.knosys.2014.08.011

Rahman, M. A., Islam, M. Z., & Bossomaier, T. (2014, June). Denclust: A density based seed selection approach for k-means. In *13th international conference on artificial intelligence and soft computing (icaisc 2014)* (Vol. 8468, p. 784-795). Zakopane, Poland.

Tan, P. N., Steinbach, M., & Kumar, V. (2005). *Introduction to data mining*. Pearson Addison Wessley.

Triola, M. F. (2001). *Elementary statistics* (8th ed.). Pearson Addison Wessley.

Wu, X., Kumar, V., Quinlan, J. R., Ghosh, J., Yang, Q., Motoda, H., ... Steinberg, D. (2008, January). Top 10 algorithms in data mining. *Knowledge and Information Systems*, *14*(1), 1-37.

Xiao, J., Yan, Y., Zhang, J., & Tang, Y. (2010). A quantum-inspired genetic algorithm for k-means clustering clustering. *Expert Systems with Applications*, *37*, 4966 - 4973.