Data Clustering using Genetic Algorithms*

Zheyun Feng
Michigan State University
Department of Computer Science and Engineering
fengzhey@msu.edu

ABSTRACT

This project focus on the problem of data clustering, where the similarity between instances are measured by Euclidean distance metric. Here we include the case where the data dimension, the number of clusters and the size of data set could be very large. Genetic algorithm is used due to its capability to capture the global optimality, leading to a promising empirical performance in the given environment. Two crossing set selection methods, six crossover approaches and a fine-tuning technique are applied in the projects. Our experiments also verify that the genetic algorithm is computationally efficient, and achieve comparable performance , if not better, as the most commonly used state-of-the-art works on large data set.

Keywords

clustering problem, genetic algorithms, crossover, large scale data set

1. INTRODUCTION

Clustering is the organization of a collection of unlabeled patterns, *i.e.* a vector of measurement or a point in a multidimensional space, into clusters based on their similarity. Intuitively, patterns within a cluster are more similar to each other than they are to a pattern belonging to a different cluster. Labels associated with clusters are data driven, *i.e.* they are obtained solely from the data, rather than learning from a training set as supervised classification does.

Clustering has been widely used in plenty of applications including data mining, pattern recognition, computer vision, image processing and information retrieval. Since recently many clustering algorithms have been developed. Classical clustering algorithms could be enumerated as k-means [8], nearest neighbor clustering, spectral clustering [10], self organizing map [7], fuzzy c-mean clustering [1], and the list goes on. Among them, genetic algorithm (GA), which proposed early in 1989 [5], attracts many attentions because it performs a globalized search for solutions whereas most other clustering approaches perform a localized search and thus easily get stuck at local optimalities. In a localized search, the new obtained solutions inherit the ones in the previous iteration [6]. Such examples are k-means [8], fuzzy clustering algorithms, ANNs, annealing schemes, and tabu search. Nevertheless, in GAs, the crossover and mutation operators can produce new solutions that are extremely different from from the previous iteration, that is where the global optimality comes from. Besides, GAs are also inherently parallel, making it possible to implement on parallel hardware so to speed up the computation.

Actually, GA is an evolutionary approaches, which applies evolutionary operators and a population of solutions to achieve a global optimal partition. Genetic algorithms include selection, recombination and mutation. Candidate solutions to the clustering problem are encoded as chromosomes, and then a fitness function inversely proportional to the squared error value is applied to determine the chromosomes' surviving likelihood in the next generation.

In the clustering problem, the solutions best fitting the data is then chosen according to a suitable criterion. In this project, we use a metric distance function rather than a distance matrix to measure the dissimilarity between the instances. The data attributes are converted to be numerical if it is ordinal, and are normalized using L_1 norm in each dimension. Thus each instance is embedded into a K dimensional Euclidean space. The aim of the clustering is to minimize the intra-cluster diversity, we use the mean square error as the evaluation measure.

In a genetic algorithm (GA) we use a model of the natural selection in real life [4], where an initial population of solutions called individuals is randomly generated. The algorithm produces new solutions of the population by genetic operations, such as reproduction, crossover and mutation. The new generation consists of the possible survivors with the highest fitness score, and new individuals estimated from the previous population using the genetic operations.

In this report, I am going to give a simple review on genetic algorithms specific for the clustering problem. When designing genetic algorithms for clustering problem, three keys play a significant role: The solution representation, the

*This is a course project for *CSE848-Evolutionary Computation* with the instructor of Dr. Bill Punch, Fall 2012

selection method, the crossover method and the mutation method.

The efficiency and effectiveness of the GA depends highly on the coding of the individuals. Naturally, a solution could be either a partitioning table, or a set of cluster centroids. The partitioning table actually is the cluster assignments for each instance in the data set. A cluster centroid has the same dimensions as an instance, and is estimated by averaging all the instances in the entire data set which correspond to the particular cluster. Two parent selecting ways are used in this project: a probability-based roulette wheel method and an elitist way with different crossover rate. In the roulette wheel selection, a parent is chosen to according to a probability estimated from the mean squared error. In the elitist method, a set of best solutions are accepted while the rest are dropped.

In the crossover phase, six methods are employed, including random crossover [9], centroid distance [9], pairwise crossover [4], largest partitions [4], pairwise nearest neighbor (PNN) [2], and iterative shrinking (IS) [3]. Among them PNN and IS are hybrid methods, which adopt few steps of the conventional k-means clustering algorithm. To improve the clustering performance for the first four crossover methods, it is meaningful to produce new solutions by crossover and then fine-tuned by a partial k-means algorithm.

The rest of the paper is organized as follows. In Section 2 we simply summarized the clustering problem. In Section 3, essential features of the genetic algorithm will be reviewed. Besides, results of the experiments are reported in Section 4, where a comparison with other state-of-the-art clustering algorithms is concluded. Finally, we draw a couple of conclusions in Section 5.

2. CLUSTERING PROBLEM

In the clustering problem, $X = \mathbf{x}_1, \cdots, \mathbf{x}_N{}^T \in \mathcal{R}^{N \times D}$ denotes an instance matrix, which consists of N examples and each instance vector has D attributes. $C = c_1, \cdots, c_N{}^T \in [1, M]^{N \times 1}$ is a vector denoting the cluster assignments. A specific clustering or partition can be expressed as $\pi = C_1, \cdots, C_M$ of X.

For a specific clustering, each cluster $C_i, i \in [1, M]$ has an alternative representative element called centroid, denoted as c_i, and it can be computed from the instance matrix X:

$$c_i = \frac{\sum_{\mathbf{x} \in C_i} \mathbf{x}}{|C_i|} \tag{1}$$

where $|C_i|$ is the number of instance corresponding to cluster C_i, which means c_i is the nearest centroid for these instances.

Given that the data are embedded in an Euclidean space, the distance between two instances $\mathbf{x}_i$ and $\mathbf{x}_j$ can be defined by:

$$d(\mathbf{x}_i, \mathbf{x}_j)^2 = \|\mathbf{x}_i - \mathbf{x}_j\|_2^2 = \sum_{k=i}^{D} (\mathbf{x}_i(k) - \mathbf{x}_j(k))^2 s \tag{2}$$

Then we can draw the mean square error for a specific clustering π as:

$$MSE(\pi) = \frac{1}{ND} \sum_{i=1}^{N} \|\mathbf{x}_i - f(\mathbf{x}_i)\|_2^2 \tag{3}$$

where f is a mapping giving the closet centroid in solution π for instance $\mathbf{x}_i$. And essentially, the problem of clustering is to determine a clustering π^* such that

$$\pi^* = \arg\min_k MSE(\pi_k)$$

When we use (2) as the distance measure, we assume that each dimension of an instance is numerical and has the same scale. So before clustering, we need to preprocessing the original data. We binarize the ordinal value, and normalize each dimension using L_1 norm. The MSE formulated in (refmse) measures the distortion of a clustering solution. This is one of possible evaluation criterion, and the user can switch the choice based on the specific environment.

3. GENETIC ALGORITHMS

A sketch of genetic algorithm is shown in Algorithm 1. The genetic algorithm evolves a population of candidate solutions represented by strings of a fixed length. Each individual of the population stands for a clustering of the data, and it could be either a vector cluster assignments or a set of centroids. An individual is initially created by randomly selecting M instances from the data set as cluster centroids and then mapping all the instances in the data set to their nearest centroid, according to (2). In each iteration, the best S_b solutions are selected to survive to the next generation. The rest of the population is replaced by new solutions generated in the crossover phase.

Algorithm 1 Outline for a genetic algorithm

```
Generate S randomly solutions for the initial generation.
while iteration times < T or termination criteria not meet do
    Select S_b surviving solutions for the next generation.
    Select solutions to form the crossing set.
    Select S_c pairs of solutions from the crossing set.
    Perform crossover to generate new solution.
    Perform mutation to the solutions.
end while
```

3.1 Solution representation

A solution to the clustering problem can be represented either by a partitioning table or a set of cluster centroids. These two depend on each other, so if we fixed one of them, the other could be uniquely constructed using the way described as follows:

- Given a set of cluster centroids, a partition table can be constructed by assigning each instance in the data set its closest centroid in respect to the distance function.
- Given a partition table, the cluster centroid can be obtained by taking the mean vector of the instance corresponds to that cluster, and the centroids for all clusters form a cluster representative.

These two variations of clustering representation give rise to two alternative approaches to the clustering problem: centroid-based and partition-based [4] clustering methods.

In the partition-based methods, the partitions are the individuals of the population, and gene (elementary unit) is a membership to a certain single cluster. The centroids thus can be computed using the way above. The partition-based clustering is wildly used in conventional clustering algorithms, and is still commonly used now but with assistant mechanisms to guarantee its performance and efficiency.

In the centroid-based approaches, the sets of centroids are the individuals of the population, and the gene is the centroid of a corresponding cluster. This way is commonly used in vector quantization. The partition table, however, is necessary when we do further operation such as crossover, and when the clustering relationship needs to update, where the nearest neighbor way can be used. And these sets of centroid is also called codebook. The generation of codebook, or in other word, the relationship between centroids and partition table is depicted in Figure 1.

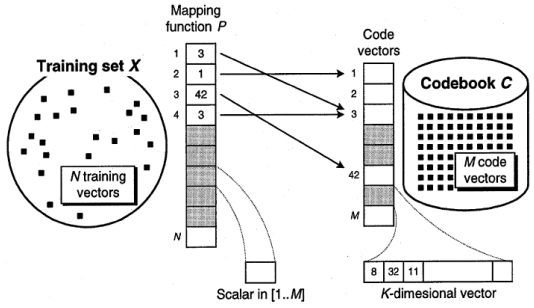


Figure 1: Relationship between partition table and codebook.

From the aspects of genetic algorithm, the difference between the two representation ways relies on the objects of genetic operations, leading to extremely different crossover and mutation strategies. Though commonly used, the drawback of partition-based representation lies that when a simple random crossover operation performed, the clusters become non-convex, specifically, the instances far away to each other may be assigned to the same cluster. When the population is large enough, theoretically this convexity won't reduce the clustering performance but greatly decrease the convergence speed. However, this can be relieved by the fine-tuning phase. In the other hand, it is impossible for the centroid-based representation to detour the partitioning, thus it would be predictably slow since the estimation of partition table has a complexity of $O(NM)$. It is therefore more reasonable to operate with the cluster centroids with assistance of the partition table. In this project, we use the centroids as the gene of solution, but cooperating with the partition table to ameliorate the efficiency and performance.

3.2 Selection method

Selection method restricts the parents by whom the new generations are reproduced, which concludes three steps: selecting the survivors, creating crossing set, and picking the parent pairs for crossover out of the crossing set. The survivor selection is usually easy, by taking the best S_b solution, where S_b is a pre-defined parameter. So our problem lies in the generation of crossing set, as well as the parent pairs for crossover. In this project, we applied two selection methods, including *Roulette wheel selection* and *Elitist selection*.

Roulette wheel selection is a probability based method, where the crossing set consists of all the solutions in the current generation, but each solution is chosen to form a pair of crossover parents with a different probability. To maintain the stability of population size, we chose $S - S_b$ pairs of parent solutions. The solution π_i is chosen with such a probability:

$$p(\pi_i) = \frac{w(\pi)}{\sum_j = 1^S w(\pi)} \tag{4}$$

where $w(\pi)$ is the weight for solution π, and it can be defined as:

$$w(\pi) = \frac{1}{1 + MSE(\pi)}$$

Elitist selection uses zigzag scanning among the best n parents, which forms the crossing set. All the solutions in the set are crossed with each other, so the number of new solution produced by the elitist selection is:

$$S_c = C_n^2 = \frac{n(n-1)}{2}$$

The value of S_b is determined by the crossover rate p_c that $S_b = (1 - p_c) \times S$. Optimal crossover rate usually lies in $p_c \in [0.8, 0.95]$, but sometimes it may occurs around 0.6. The value of the crossover rate p_c can be determined by cross-validation. In the project, we used three selection strategies:

- Roulette wheel selection, with $S_b = 5$, and then choosing $S - S_b$ pairs of parents, and each solution is chosen with a probability based on the fitness function.
- Elitist selection 1, with crossover rate equal to 0.8, where $S_b = 9$, and $n = 9$.
- Elitist selection 2, with crossover rate equal to 0.9, where $S_b = 5$, and $n = 10$.

3.3 Crossover operators

The crossover is performed in order to create a new better solution from the two selected parent solutions, denoted here by A and B and each of them has M centroids. Since we used the centroid-based cluster representation, the crossover in this project is thus be designed particularly with crossover operations on set of centroids. Essentially, the crossover phase can be regarded as selecting M cluster centroids from the $2M$ two parent solutions. The newly generated solution could be completely different from its parents, inheriting only partial parent information, and this is the underlying reason that genetic algorithm is capable to do global search. In consequent, we simply go over six adopted crossover methods: random crossover, centroid distance, pairwise crossover, largest partitions, pairwise nearest neighbor, and iterative shrinking. In the crossover, only the cluster centroids are produced to form a new solution, the data instance are necessary to partition to them using the nearest neighbor criterion.

Random crossover [9] is performed by just randomly picking $M/2$ centroids from each of the two parents, while duplicate centroids are rejected and replaced by repeated picks. Random crossover is simple and efficient, since there is no extra operations but randomly selection. But for each generation, it is not highly robust since it is totally possible to pick up a bad set of centroids. Although this could be compensate by more iterations, it will certainly prolong the evolution time.

Centroid distance [9] first computes the centroid of the entire data set, denoted as c^*. Then all the centroids in both parents are sorted in respect of their distance to c^*. The produced solution consists of $M/2$ centroids from parent A, which are the closest to c^* in A, and another c^* centroids from parent B, which are the ones furthest away from c^* in B.

Pairwise crossover [4] pairs the centroids of the two parents based on the nearest neighbor criterion using greedy manners. Then crossover is performed by randomly taking one cluster centroid from each pair of centroids. When two centroids in parent A have the same nearest neighbor in parent B, this neighbor will be coupled with the one closer to it. Empirically, this algorithm does not give the optimal pairing but it is a reasonably good heuristic for the crossover purpose.

Largest partitions [4] pickes the M cluster centroids based on a greedy heuristic based assumption that the larger clusters are more important than the smaller ones, because eventual purpose for evolutionary clustering is to minimize the intra-cluster diversity. The cluster centroids thus should be assigned to a large concentration of data instances. In this method each cluster in both parent A and B is assigned with a value equal to the cluster size in its own solution, and then these $2M$ centroids are union together. Finally the new solution is formed by the M centroids with the largest number of associating data instances.

Pairwise nearest neighbor [2] regards the crossover phase as a clustering problem. It first combines the cluster centroids from both parent A and B, then eliminates the duplicated centroids from the obtained union, and finally iteratively performs a PNN clustering on this union until the number of clusters in the union is exactly equal to M. Then this M newly clustered centroids are the new solution we desired.

The PNN clustering algorithm starts by initializing a clustering of size $2M$ where each instance is considered as a individual cluster. Then two clusters with the shortest pairwise distance would be merged at each step, and this repeats until the number of clusters is equal to M.

The main restriction of the PNN method is that the clusters are always merged as a whole. Once the instances have been assigned to the same cluster, it is impossible to separate them later. This restriction is not significant at the early stage of the process when merging smaller clusters but it can deteriorate the clustering performance at the later stages when merging larger clusters.

Iterative shrinking [3] first unions the centroids from both parent A and B similarly as PNN, then generates the partition by a sequence of cluster removal operations: one clusters are removed at a time by reassigning its corresponding instance to the nearest remaining clusters. PNN is a special case if IS.

IS first finds the second nearest cluster for each data instance in the selected cluster, and forms a secondary partition $Q = q_1, \cdots, q_N | q_i \in [1, M]$ for every data vector, according to a modified distance:

$$q_i = \arg \min_{1 \le j \le M, j \ne p_i} \frac{n_j}{1 + n_j} \|\mathbf{x}_i - c_j\|_2^2$$

where n_j is the number of instance whose second closest centroid belongs to cluster j. Then the cluster whose removal gains the least increasing cost will be removed, in other word, cluster a in the uion will be removed if its removal cost is the least. The removal cost is defined as follows:

$$d_a = -\sum_{j=1}^{M} |s_{a,j}| \cdot \|c_a - c_{a,j}\|_2^2 + \sum_{j=1}^{M} \frac{n_j |s_{a,j}|}{n_j + |s_{a,j}|} \|c_j - c_{a,j}\|_2^2$$

where $s_{a,j}$ is a subcluster of cluster s_a, which is defined according to the secondary partition Q:

$$s_{a,j} = \{\mathbf{x}_i \in s_a | q_i = j\}$$

and $|s_{a,j}|$ is its cardinality. Correspondingly, $c_{a,j}$ is the centroid of subcluster $s_{a,j}$.

Once a cluster s_a with the smallest removal cost is eliminated, its associated instance should be partitioned to a rest cluster, and this updating is defined as:

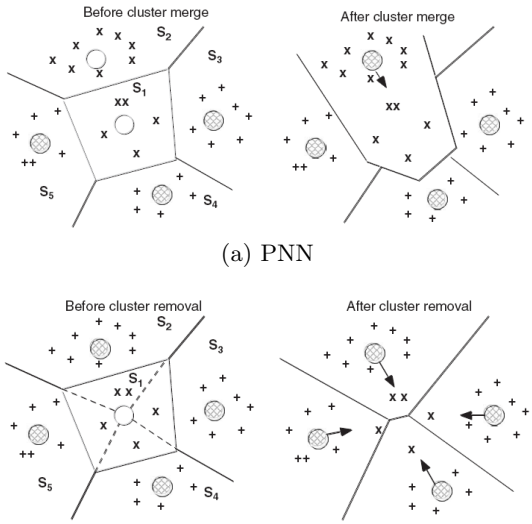
$$\forall \mathbf{x}_i \in s_a : \; p_i \leftarrow q_i.$$

And this removal continues until the number of left cluster is exactly equal to M.

IS versus PNN:
The PNN method can be seen as a special case of the IS method, as it can be simulated by the IS method as follows. We first select the cluster to be removed as one of the two clusters (s_a and s_b) selected for the merge. The merge is then performed by moving all the instances from s_b to s_a, and thus, removing s_b. The centroid of s_a is updated accordingly. The result is equivalent to that of the PNN method, and it is easy to see from Figure 2 that some of the instance reassignments could be done better resulting in a smaller increase in the cost function value.

The difference of the merging and removal strategies is illustrated further in Figure 2. We have six data instances located symmetrically, and the task is to find a partition of two clusters. After the first three merges, the output of the PNN and IS methods are equivalent but in the fourth merge the PNN method is already restricted by the previous merges and the result is suboptimal. The IS method, on the other hand, ends up with the optimal result no matter what is the order of the previous cluster removals.

It is noted, that it is still possible (although rare) to get better result by the PNN method than by the IS method because locally optimal steps does not necessarily lead to



(a) PNN

(b) IS

Figure 2: The cluster removal process of PNN and iterative shrinking.

the global optimum. Nevertheless, it is expected that the IS method would give better partition than the PNN method in most cases.

3.4 Mutation

Each cluster centroid is replaced by a randomly chosen data instance with a probability p_m, which is also called mutation rate. This operation is performed before the partition phase. In this project, we fixed the mutation rate $p_m = 0.05$, which is obtained through cross-validation and empirically yields a stable but good performance.

Mutation rate is a leverage between diversity and stability. Ideally, we want a larger diversity and a higher stability, so we need to make a good compromise between the two and well control the mutation rate. Commonly, an optimal mutation rate existing in the range of $p_m \in [0.005, 0.01]$.

3.5 Fine-tuning

One can try to improve the algorithm by applying a few steps of the k-means algorithm for each new solution. The crossover operation first generates a rough estimate of the solution which is then fine-tuned by a partial k-means algorithm. This modification allows faster convergence of the solution than pure genetic algorithm.

The implementation of fine-tuning technique in this project is described as follows. The initial solution is iteratively modified by partitioning according to centroids, and computing centroids from the newly obtained partition table. In the first phase the centroids are fixed and the clusters are recalculated using the nearest neighbor criterion. In the second phase the clusters are fixed and new centroids are estimated.

4. EXPERIMENTAL RESULTS

We evaluate clustering quality by both the mean square error and the convergence rate.

4.1 Data set description

We applied our sequence of genetic algorithms to 6 data sets from the UC Irvine repository [1] The summary of the statistics about the data sets is presented in Table 4.1.

Table 1: Data sets and their statistics

Data set	Attris	Size	Clusters	Type
iris	4	150	3	real
abalone	10	4177	29	categorical,real
wine	11	4198	7	integer, real
letter	16	20000	26	integer
segment	18	2310	7	real
network	21	53413	24	integer, real

The types of data are various, and different attributes may have values of different types, range, so we need to do some pre-processing before clustering. We need to convert the categorical attributes to binary ones, for example, for an attribute indicating sex (Female or Male), we use 1 to represent female, and 0 to represent male. Then we normalized the data into range of $[0, 1]$ using L_1 norm in respect to columns, making each attribute having the same contribution to the distance between data.

All tests have been performed in the server of *winry.egr.msu.edu*, which is equipped with Sun Fire x4150 with dual Inter Xeon X5460 3.16GHz 64 bit processor, 64GB RAM and Linux operating system.

Other experiment setup are explained in Section 3.

4.2 Effect of fine-tuning

Fine-tuning is wildly used to improve the performance of crossover operation. In the clustering problem, the fine-tuning in this project mentions to a few step of k-means algorithm. The performance of the genetic algorithm with or without fine-tuning is illustrated in Figure 3 as a function of the number of generations for both *wine* and *segmentation* data sets.

For random crossover, centroid distance, pairwise crossover and largest partition, the inclusion of fine-tuning is essential; even the worst candidate in each generation is better than any of the candidates without k means, making a great improvement in the performance. Besides, with the fine-tuning, random crossover, centroid distance and pairwise crossover even converges much faster. For PNN and IS, there is also some improvement in the performance, although not significant. For the *segmentation* data set (bottom) in Figure 3, this improvement is apparent[2].

The implementation time for all six crossover algorithms with or without fine-tuning on *wine* is presented in Table 4.2.

[1]The UC Irvine repository data can be downloaded from `http://archive.ics.uci.edu/ml/`.

[2]Since the fine-tuning will modify the clustering performance greatly, alleviating the influence of other conditions, we don't use fine-tuning for most cases stated later. Fine-tuning not applied if not noted.

Table 2: Time cost with/without kmeans on *wine* data set.

	Random crossover	Centroid distance	Pairwise crossover	Largest partition	Pairwise nearest neighbor	Iterative shink
with	16.2935	15.7701	16.3933	17.6014	32.8162	418.9808
without	2.2258	2.0688	2.6190	3.8513	16.5585	422.2040

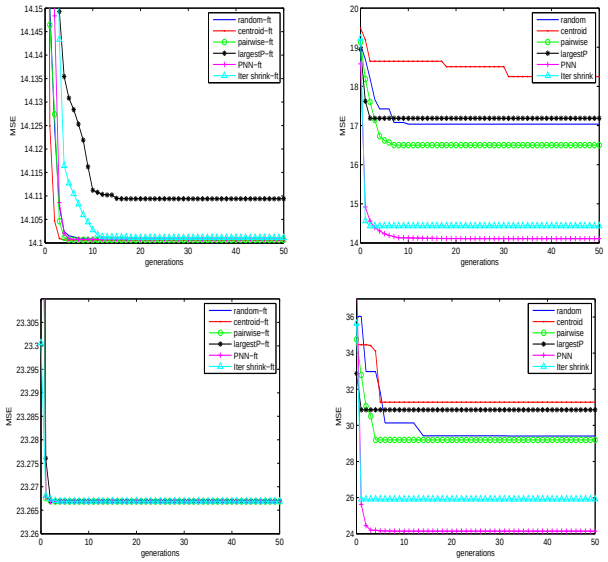


Figure 3: Convergence of the various crossover algorithms for both *wine* (top) and *segmentation* (bottom)data set eith with kmeans(left) or without kmeans (right), the elitist selection method was used.

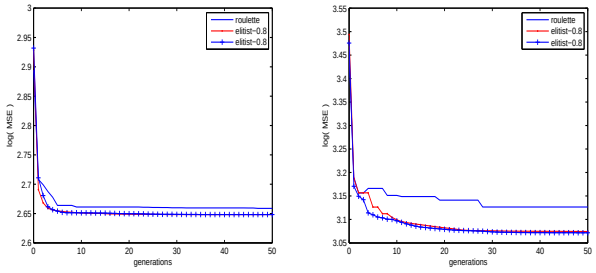


Figure 4: Performance comparison of the three selection strategies described in Section 3.2, where PNN crossover is performed on both *wine* (left) and *letter* (right) data sets.

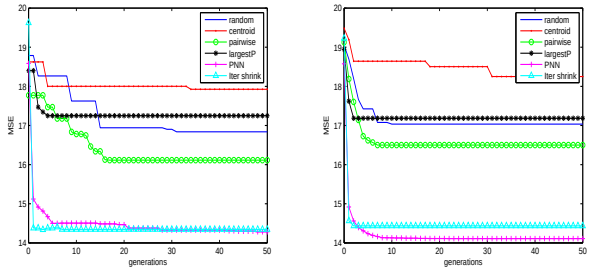


Figure 5: Performance comparison of the two selection strategies: roulette wheel selection(left), and elitist selection(right), with different crossover operators on *wine* data set.

The drawback of the hybridization with fine-tuning is that the running time considerably grows. Fortunately, it is not necessary to perform the k means algorithm to its convergence but only a couple of steps (two in the this project) suffice. The results are similar for the other data sets not shown in Table 4.2.

4.3 Effect of selection method

The performance of the different selection strategies is summarized in Figure 4.3, where PNN crossover is on both the *wine* and *letter* data sets. The experimental strategies are described in Section 3.2: The roulette wheel wheel selection, elitist selection with crossover rate = 0.8, and elitist selection with crossover rate = 0.9. The selection method seems to have a smaller effect on the overall performance. In most cases of PNN the elitist selection are better than the roulette wheel selection, not only for the performance but also for the convergence rate; and the gap is quite large for *letter* data set.

The comparison of roulette wheel selection and elitist selection for different crossover operations are shown in Figure 4.3. For random and pairwise crossover, roulette wheel clearly outperforms elitist selection; while for PNN elitist is slightly better. For largest partition and iterative shrinking, the two selection methods have no large difference.

4.4 Effect of crossover

The performance of the different crossover methods is illustrated in Figure 4.4 as a function of the number of generations. And the overall performance can be referred Section 4.7: MSE comparison in Table 4.7 and running time in Table 4.7.

PNN and IS are well outperforms the other operations by giving the best clustering with the fewest number of iterations; and in most cases, pairwise works better than the rest three. IS converges the fastest among all, and its performance is comparable to PNN that sometimes PNN wins and sometimes the contrast. Usually, with roulette wheel selection, IS performs better; while with the elitist selection PNN wins. Pairwise also converges very fast. Mostly, centroid crossover has the worst performance and converges the slowest.

4.5 Effect of mutation rate

The performance of the different different mutation rate is illustrated in Figure 7 as a function of the number of gen-

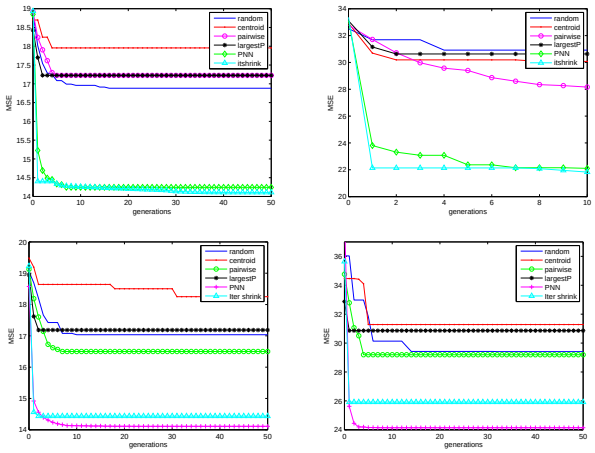


Figure 6: Convergence of the various crossover algorithms for *wine* (left column), *letter* (top right) and *segmentations* (bottom right) data sets. Top row uses roulette wheel selection and bottom row uses elitist selection.

erations. Among the other parameters, the amount of mutations had only a small effect on the performance, if the mutation is well chosen and it can be obtained through cross validation.

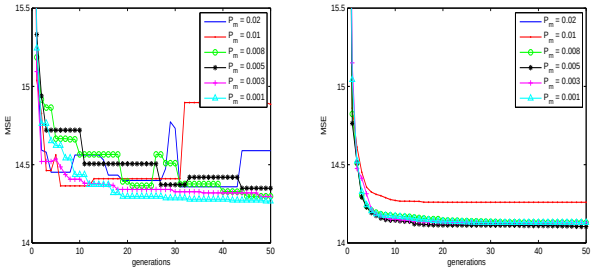


Figure 7: Performance comparison of mutation rate, with roulette wheel selection + PNN (left), and elitist selection + IS(right), on *wine* data set.

From previous observation, we know that IS works better with roulette wheel selection than with elitist selection, see the left column in Figure 4.4 also for *wine* data set. And it also shows that PNN works better than IS in term of MSE. But in Figure 7, the observation is surprising that when combing these selections and crossovers, elitist selection + IS well outperforms together roulette wheel selection + PNN.

The purpose of introducing mutation operation is to increase the versatility and diversity of the solution group, expecting to produce genius solution which yields an extraordinarily clustering performance. However, this probability is rare, and most of the mutation reduces the performance. When the mutation occurs at the best solution in a generation, very possible, this best solution would be sub-optimal, making the best solution in the current generation even worse than the one in previous generation, justified by the left example in Figure 7. In this project, we see that when this situation happens, the convergence curve would be no more smooth, indicating that the stability of evolution is in threaten.

So for the unexpected phenomenon in Figure 7, both the elitist selection and IS itself reduce the solution diversity too much, either by reducing the number of parent for crossover, or by over-tuning. However, mutation can compensate this by introducing variation in the population, leading to the situation that combining two conditions resulting worse results gives better performance.

4.6 Conclusions of GAs

The above observations demonstrate two important properties of genetic algorithms for large scale clustering problems. A successful implementation of GA should direct the search **efficiently** but it should also retain enough genetic **variation** in the population. The first property is clearly more important because all ideas based on it, including of fine-tuning, PNN crossover, IS crossover elitist selection, give good results. Their combination, however, reduces the genetic variation so that the algorithm converges too quickly, as shown in Figure 7, where IS makes a good example that when using elitist selection together with iterative shrinking, the corresponding converges extremely fast but performs usually worse than PNN. So to some extent, IS and elitist selection have well taken advantage of the first properties. Thus, we can use the roulette wheel selection or mutation to compensate the loss of the genetic variation.

An interesting but less important question is whether extra computing resources should be used to increase the generation size or the number of iteration rounds. Additional tests have shown that the number of iteration rounds has a slight edge over the generation size but the difference is pretty small and the quality of the best clustering depends mainly on the total number of candidate solutions.

4.7 Comparison with other clustering algorithms

GA is next compared to other existing commonly used clustering algorithms, including k-means [8], spectral clustering [10], self organizing map [7], fuzzy c-mean clustering [1][3]. The best performance results for GAs, k-means, self organizing map, fuzzy c-mean clustering and spectral clustering are summarized in Table 4.7 in term of MSE, and in Table 4.7.

We observe that GA clearly outperforms the other algorithms used in comparison. Note that here GA is not tuned to be the best, and the results are just the commonly running results. The statistics show also that GA is relatively independent on the initialization whereas the results of k-means have much higher variation.

The drawback of GA is its high running time, and when the data size increases, this cost could be prohibitive. For GAs, higher quality clusterings are thus obtained at the cost of larger running time.

5. CONCLUSIONS

GA solutions for large scale clustering problems were studied in this project. The implementation of a GA-based clustering algorithm is quite simple and straightforward. However,

[3]The baselines are implemented using the Matlab toolbox.

Table 3: Mean square error for all methods

Data sets	Random crossover	Centroid distance	Pairwise crossover	Largest partition	PNN	Iterative shinking	Kmeans	SOM	Fuzzy c-means	Spectral
iris	32.05	34.62	34.17	36.68	**29.16**	**29.16**	**29.16**	**29.16**	29.51	30.25
segments	30.92	30.07	28.16	30.63	22.11	**21.83**	23.27	23.27	24.16	23.44
abalone	3.013	3.342	2.660	3.664	**2.051**	2.150	2.078	2.397	2.158	2.580
wine	29.07	33.20	28.67	29.49	**23.27**	25.11	24.21	24.22	28.71	24.50
letter	30.92	30.07	28.16	30.63	22.11	21.83	21.50	**21.48**	50.70	23.29
network	4.60	4.94	4.10	5.44	3.58	3.40	**3.16**	3.17	6.16	3.71

Table 4: Time cost for all methods

Data sets	Random crossover	Centroid distance	Pairwise crossover	Largest partition	PNN	Iterative shinking	Kmeans	SOM	Fuzzy c-means	Spectral
iris	0.194	0.241	0.341	0.315	0.525	1.532	0.014	2.035	0.028	0.050
segments	0.983	0.966	1.416	1.368	5.27	80.61	0.065	3.728	1.283	0.281
abalone	12.26	12.90	12.84	13.93	53.00	118.9	1.894	10.44	6.118	1.019
wine	1.891	2.024	2.598	2.264	12.48	186.6	0.270	4.644	2.006	1.409
letter	26.78	26.14	27.87	31.98	183.2	21142	5.288	56.13	38.49	8.409
network	140.1	141.2	143.6	170.9	599.4	5223.3	15.3	157.4	124.8	14.35

problem specific modifications were needed because of the nature of the data. Euclidean distance metric is not good at capturing elliptical or chain-like clusters, so distance metric learning could be adopted to improve the performance. Also, better fitness function could be used, maybe the entire or partical label information can also be used to help improve the performance. New candidate solutions are created in the crossover but they are too arbitrary to give a reasonable solution to the problem. Thus, the candidate solutions must be fine-tuned by a small number of steps of the k-means algorithm.

The main parameters of GA for the clustering problems studied here are the inclusion of k-means steps, and the crossover technique, as well as the analysis of mutation rate, which reflects the essence of evolutionary approaches like GA. In most cases, the mutation probability and the choice of selection method seem to be of minor importance. The centroid-based representation for the solution was applied. The results were promising for this configuration of GA. The results of GA, which is measured by the intracluster diversity in this project, were better than those of k-means and other common clustering algorithms. Totally SOM gave competitive results to GA with much more computing efforts.

6. REFERENCES

[1] J. C. Bezdek. Fuzzy c-means cluster analysis. *Scholarpedia*, 6(7):2057, 2011.

[2] P. Fränti. Genetic algorithm with deterministic crossover for vector quantization. *Pattern Recogn. Lett.*, 21(1):61–68, Jan. 2000.

[3] P. Fränti and O. Virmajoki. Iterative shrinking method for clustering problems. *Pattern Recogn.*, 39(5):761–775, May 2006.

[4] P. FrÃd'nti, J. KivijÃd'rvi, T. Kaukoranta, and O. Nevalainen. Genetic algorithms for large scale clustering problems. *Comput. J*, 40:547–554, 1997.

[5] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[6] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Comput. Surv.*, 31(3):264–323, Sept. 1999.

[7] T. Kohonen, M. R. Schroeder, and T. S. Huang, editors. *Self-Organizing Maps*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 3rd edition, 2001.

[8] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. L. Cam and J. Neyman, editors, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[9] C. A. Murthy and N. Chowdhury. In search of optimal clusters using genetic algorithms. 17(8):825–832+, 1996.

[10] A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*, pages 849–856. MIT Press, 2001.