Final Year Project

Design Report

Barry Juans (10216709)
Supervisor: Prof. Steven Guan

1. Summary of Proposal

1.1. Project Specification

Clustering problem is a problem of categorizing similar dataset instances into groups [1]. The main idea of clustering is to form clusters which are heterogeneous, while at the same time maintaining homogeneity among the in-cluster elements. In this research, the similarities will be calculated based on distance of dataset's attributes.

Many clustering algorithms such as k-means [2], hierarchical clustering [3], and OPTICS [4] were invented to solve the clustering problem, and each of them has their own strengths and weaknesses. However, despite the amount of researches done concerning clustering algorithms, clustering data in high dimensional space still poses a huge challenge, since in high dimensional space, the definition of similar or dissimilar is blurred. Such phenomenon is called the 'Curse of Dimensionality' [5].

Several evolutionary algorithms have also been introduced to deal with clustering problem; one of them is Genetic Algorithm (GA). Unlike most clustering algorithms, GA's concept is not derived from statistics; rather, it borrows the idea of natural selection [6]. GA works by iteratively forming possible solutions and then selecting the best solution among them, and if the selected solution is better than the solution obtained from previous iteration, then the old solution is replaced with the new one. The whole process is repeated until a termination condition is reached. Through these processes, GA is predicted to produce an optimal solution.

In the experiment, discriminant hyperplane, similar to the one used in Perceptron [7], will be used to separate one cluster from the rest of dataset instances. The rationale behind it is that if the discriminant hyperplane can be used for classification problem, it could be used for clustering problem as well [8]. In the algorithm, GA will find the optimal hyperplanes to separate clusters.

The aim of research is to develop a GA-based clustering algorithm which could form clusters accurately.

1.2. Changes to Original Specification

In the original specification, the algorithm is expected to be purely based on GA; however, experiments have shown that using GA alone is not possible to form accurate clusters.

One issue with the original design is that discriminant hyperplanes are useful only for datasets which are linearly separable. For linearly inseparable datasets such as the Iris dataset [9], GA will tend to merge two similar clusters together. To resolve this issue, a new layer of discriminant hyperplanes, similar to the hidden layer used in Multilayer Perceptron [10], could be added to the existing model; however, this would significantly slow down the whole GA process as GA would also need to find the optimal hidden layer discriminant hyperplane for each of the discriminant hyperplanes in the first layer. It should be noted that GA itself is already slow in its computations and such modifications would just exacerbate the problem.

In addition, the original algorithm has difficulty finding the correct clusters when the actual number of clusters is large. Since the idea of discriminant hyperplane is to separate a cluster from the rest of dataset, the cluster formed by the hyperplane would sometimes contain the outlier elements from other clusters, simply because those outlier elements are closer to the centroid of the formed cluster as compared to the centroid of the 'rest of dataset' instances. Basically, the linear separation done by the discriminant hyperplane disregards the fact that the 'rest of dataset' instances also consist of more than one cluster.

A new method is added to the original design to resolve these problems; it is called 'split-fusion'. The 'split' part of the algorithm will divide a cluster into two smaller ones and 'fusion' will break down a cluster into single instances and then merge each instance with other closest clusters. This new algorithm is expected to be able to separate linearly inseparable data as well as reforming clusters containing elements of other clusters.

The algorithm has also been modified so that GA may perform the separation only once for each cluster, meaning that one cluster will be produced by at most one discriminant hyperplane. It is possible to develop a recursive approach in separating the cluster, meaning that after a cluster is produced, GA will perform additional separation to the produced cluster, but experiment has shown that the single-separation method, combined with split-fusion, is sufficient to produce the same clustering results. Furthermore, this approach will improve the computational time for the whole algorithm.

1.3. Research and Analysis Done

Several datasets have been obtained from UCI Repository [11] and analyzed; in addition, research papers involving other state-of-art clustering algorithms have been reviewed in order to determine the benchmark performance for those datasets [12][13]. Some Artificial Neural Network models have also been reviewed to obtain a better understanding on the usage of discriminant hyperplane.

Real world datasets obtained from the repository, such as Iris and Wine datasets, show that real world datasets are likely to contain attributes that are irrelevant to clustering process, also known as noise attributes. This highlights the importance of evaluating each attribute individually before directly applying distance calculations.

Most of the programming work has been completed and some testing has also been done to ensure that the code is working correctly. Currently, the highest priority is to optimize the algorithm so that it is capable of achieving high accuracy for the benchmark datasets as well as the artificial datasets.

2. Algorithm Design

2.1. Algorithm Structure

The algorithm *GACluster* uses the iterative model for clustering. In each loop, GA is used to find a cluster, and then split-fusion method is used to refine that cluster. The whole process is repeated until all dataset instances have been clustered. The flow of algorithm *GACluster* is as follows (Fig 1):

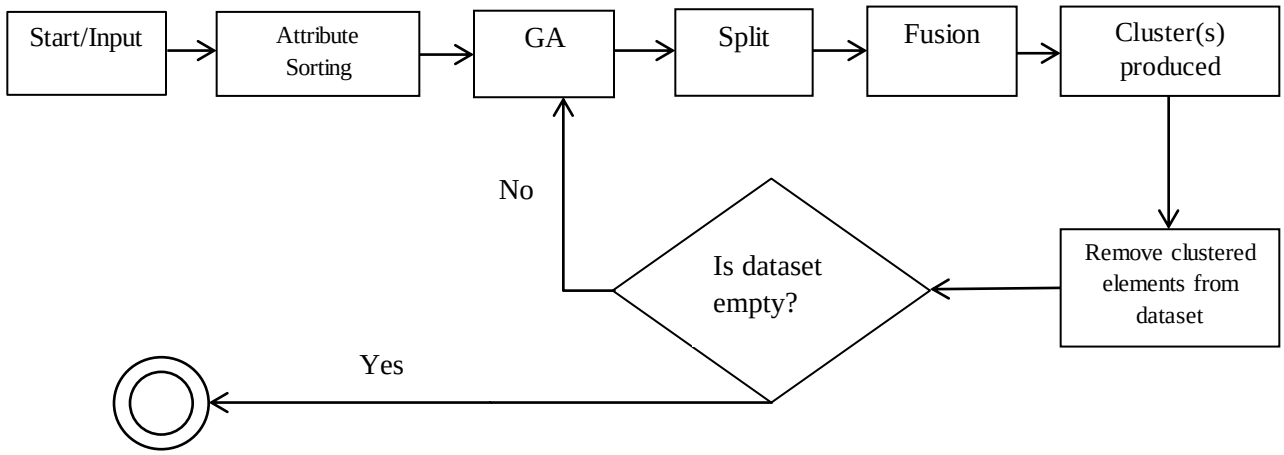

Fig 1. Flow Diagram of *GACluster* algorithm

As shown in Fig 1, all the dataset instances would have been clustered when *GACluster* terminates.

2.2. Input

The only input for *GACluster* algorithm is a dataset presented as a simple text file. The dataset itself should contain the following list of information:

1. Number of instances
2. Number of attributes
3. Attribute values for all instances
4. Optimal number of clusters
5. Correct cluster labels for all instances

4 and 5 are required in order to measure the accuracy of *ClusterGA*. In the experiment, all the information above will be formatted as such (Table 1) ($(\text{Attribute Value})_{xy}$ refers to the attribute value of x^{th} instance and the y^{th} attribute of that instance) :

Number of Attributes (in this example, there are 3 attributes)			
Optimal Number of Clusters (in this example, there are 2 clusters)			
Number of Instances (in this example, there are 4 instances)			
$(\text{Attribute Value})_{11}$	$(\text{Attribute Value})_{12}$	$(\text{Attribute Value})_{13}$	Cluster 1
$(\text{Attribute Value})_{21}$	$(\text{Attribute Value})_{22}$	$(\text{Attribute Value})_{23}$	Cluster 2
$(\text{Attribute Value})_{31}$	$(\text{Attribute Value})_{32}$	$(\text{Attribute Value})_{33}$	Cluster 2
$(\text{Attribute Value})_{41}$	$(\text{Attribute Value})_{42}$	$(\text{Attribute Value})_{43}$	Cluster 1

Table1. Format of Dataset Input

2.3. Data Structures

The algorithm will mostly use Vectors to store lists of values or objects. Vector is chosen because its dynamic capacity and it allows for quick insertion and deletion of data. After a dataset has been selected as input, the algorithm immediately converts the whole dataset to a *PatternsPool* class, with each of dataset instances being an object of *Pattern* class. In the later part of the algorithm, *Cluster* class will be created to store all the *Pattern* objects which belong to the same cluster. The relationship between the 3 classes is shown below (Fig 2):

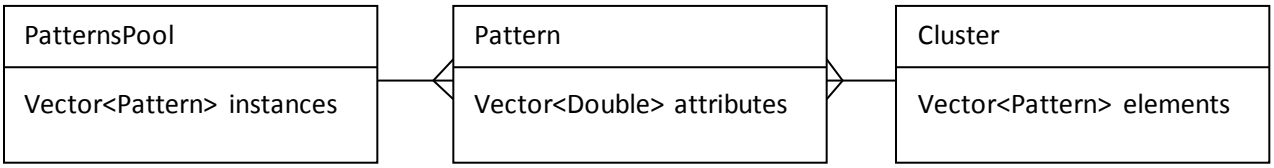

Fig 2. Class relationships between PatternsPool, Pattern and Cluster objects

2.4. Attribute Sorting

As stated previously, noise attributes are fairly common in real world datasets, and it would be difficult to discern the noise attributes from the critical ones when a cluster is already formed; therefore, the identification is done before any clustering process commences.

In this phase, the algorithm will attempt to sort the attributes according to the level of noise. The noise attribute will generally have one of the following characteristics:

1. Relatively small deviations among the attribute values (See Attribute 1 of Table 2).
2. Relatively large, but statistically random [14] attribute values (See Attribute 2 of Table 2).

Attribute 1	Attribute 2
0.1	5
0.1	-2.3
0.2	10
0.1	0.03

Table 2. 2 attribute values for 4 dataset instances. Attribute 1 values are relatively equal and Attribute 2 values are random. Both attributes are considered noise attributes.

Conversely, the critical attributes are the attributes which have relatively large deviations in values, while at the same time, follows certain recognizable patterns. Based on this knowledge, an algorithm is created to sort the attributes according to the noise level.

In the algorithm (Algorithm 1), the first loop and inner loop calculate the sum of deviations for each attribute. Each deviation is calculated based on the absolute difference between an instance's attribute value and the attribute mean; divided by the maximum attribute value. The second loop calculates the

average deviation for each attribute. The last loop and inner loop calculate the modified average deviation for each attribute. The modification is done by multiplying the average deviation by absolute value of (n-k)-k. n represents the number of instances in the dataset and k is the number of instances whose attribute value is below the mean; (n-k) is simply the number of instances whose attribute value is above or equal the mean. If the value of |(n-k)-k| is low, it implies that the attribute values are relatively random; it shows that there is no noticeable pattern in the dataset. Conversely, high |(n-k)-k| value implies that there is certain pattern existing in the attribute, which causes the value of (n-k) to be relatively different from k. After the modified average deviations have been calculated, the attributes are then sorted based on those values; lower values imply that the attributes are less important. Note that this technique is more effective when number of optimal clusters is odd or cluster sizes are uneven; otherwise, there is a possibility that |(n-k)-k| is equally low for noise and critical attributes.

Algorithm: Attribute Sorting

```
int n = number of instances;
int d = number of attributes;
int k = 0; //used for calculations
Vector<Pattern> instances; //Vector containing all dataset instances
Double value = 0; //used for calculations of deviations
Pattern centre; //Centroid of all dataset instances
Vector <Double> temp; //Vector to store results of calculations

for i = 1 to d
    max = maximum value of attribute i among all instances;
    for j = 1 to n
        value += (Absolute difference between the attribute(i) value of instances(j) and centre)/max;
        //find the normalized deviation to centre value
    end
    Append value to Vector temp;
    value = 0; //re-initialize value for the next attribute
end

for i = 1 to d
    temp(i) = temp(i)/n; //calculate average deviations for each attribute
end

for i = 1 to d
    max = maximum value of attribute i among all instances;
    k = 0; //initialize k value
    for j = 1 to n
        value = attribute(i) value of instances(j);          //find the get the instance's attribute value
        if(value < mean of attribute(i)) k++; //find the number of times attribute value is lower than mean
    end
    temp(i) = temp(i)*(Absolute value of ((n-k)-k)); //modify the average deviation based on (n-k)-k
    value = 0; //re-initialize value for the next attribute
end

Sort the attributes based on Vector temp values //high temp value means the attribute is critical
```

Algorithm 1. Identify Critical Attributes

2.5. Genetic Algorithm (GA)

The main function of GA in the algorithm is to find the optimal discriminant hyperplane which could separate a set of similar instances from the rest of dataset.

2.5.1. Encoding of Discriminant Hyperplane

Each discriminant hyperplane has a weight vector of size d, with d being the number of attributes/dimensions of the dataset, plus a constant c and a Boolean variable b. The discriminant hyperplane's function is similar to Neural Network's Perceptron's. As an example, for an instance defined as $[a_1, a_2, a_3 \dots a_d]$ and the discriminant defined as $[l_1, l_2, l_3 \dots l_d, c, b]$, the calculation will be:

$$Z = a_1 * l_1 + a_2 * l_2 + a_3 * l_3 \dots + a_d * l_d + c \qquad \textbf{(Eq 1)}$$

If the value of Z is larger than zero and the Boolean value b is positive (true), then the instance is placed into the ACCEPTED set, meaning it is considered part of the cluster; otherwise it is placed into the REJECTED set, or the 'rest of instances'. Conversely, if the Boolean value is negative (false) and Z is larger than zero, the instance will be placed in REJECTED set [15]. The following graph shows the structure of the discriminant hyperplane (Fig 3):

Weights				Constant	Boolean
w_1	w_2	$w_{3\dots}$	$\dots w_d$	c	b

Fig 3. Structure of Discriminant Hyperplane

Each weight values range between -1 to 1 and for the experiment; there is a 40 percent chance that a weight value would be set to zero. The constant c is a random value between the minimum and maximum attribute value of all the attributes.

2.5.2. Fitness Function

GA uses fitness function in order to evaluate the discriminant hyperplane. In this experiment, the fitness function calculation is based on the concept of majority voting. For each instance, the fitness calculation is:

$$y = \sum_{i \in ACC} drej_i / (drej_i + dacc_i) \qquad \textbf{(Eq 2)}$$

y is calculated using (Eq 2) if majority of attributes are closer to centroid of ACCEPTED cluster; otherwise:

$$y = -(\sum_{i \in REJ} dacc_i / (drej_i + dacc_i)) \qquad \textbf{(Eq 3)}$$

dacc refers to the absolute distance between the instance's attribute and ACCEPTED centroid's attribute. Conversely, *drej* is the absolute distance between the instance's attribute and REJECTED centroid's attribute. The value of *dacc* is inversely correlated with value of *drej*.

ACC in here refers to the set of attributes that are closer to the centroid of the ACCEPTED cluster and i refers to one of the ACC attributes; therefore, if the majority of attributes of an instance are closer to the ACCEPTED, y will be positive, otherwise it will be a negative value. Since the algorithm needs a fitness function for the cluster and not for each instance, the cluster fitness is defined as:

$$\text{fitness} = \sum_{i \in ACCEPTED} y_i \qquad \textbf{(Eq 4)}$$

In (Eq 4), i refers to an instance belonging to ACCEPTED set. Basically, fitness is the sum of fitness of each instance in the ACCEPTED cluster.

It should be noted that if there is no instance in the ACCEPTED set, the fitness value is 0; and if there is no instance in REJECTED set, the fitness value is 0.01;

2.5.3. GA Processes

GA has several processes to go through before getting the optimal discriminant hyperplane; they are:

1. Initialization
2. Selection
3. Crossover
4. Mutation
5. Evaluation

In GA, each discriminant hyperplane will be referred to as 'chromosome'. As stated previously, a cluster will be defined by only one chromosome. This approach is similar to Michigan approach [16].

In the initialization stage, n chromosomes will be randomly generated and stored, in which n is the pre-determined population size. In addition, the fitness value for each chromosome will also be calculated and stored. (Check Table 3 in the next section for the Main Population Size set in the experiment)

In selection phase, each fitness value in the main population is divided by the sum of population's fitness values to obtain the normalized fitness values. The chromosomes are then sorted in descending order according their normalized fitness values. The vector containing the accumulated normalized fitness values are then created, in which the accumulated normalized fitness is equal to the sum of a chromosome's normalized fitness plus the normalized fitness values of all previous chromosomes. As an example, accumulatedfitness[i] = normalfitness[i] + normalfitness[i-1] + normalfitness[i-2]…+ normalfitness[0]. A random number between 0 and 1 will be generated; the first chromosome whose accumulated fitness is greater than the random number will be selected and placed into the mating population. The previous 2 processes (random number generation and selection of chromosome) will be

repeated until the pre-determined mating population size has been reached. This selection process is called 'roulette wheel selection'. (Check Table 3 in the next section for the Mating Population Size set in the experiment)

In crossover phase, a discriminant in the mating population and its adjacent discriminant (chromosome[i] and chromosome[i+1]) will have their variables (weights, constant, and Boolean value) swapped with each other, depending on the crossover probability/rate. For example, if the crossover rate is 0.5, for every weight value, constant, or Boolean value of a chromosome, there will be a 50 percent chance that it will be swapped with the neighbour's corresponding value. The whole process will be repeated for all the chromosomes in the mating population. (Check Table 3 in the next section for the Crossover Rate set in the experiment)

In mutation phase, each of chromosomes in the mating population will have its variables changed randomly, depending on the mutation rate. For example, if the mutation rate is 0.4, for every variable in a chromosome, there is 40 percent chance that it will be changed to a new value. In the experiment, the new value is randomly generated, similar to initialization stage. (Check Table 3 in the next section for the Mutation Rate set in the experiment)

After the mutation phase is completed, all the chromosomes in the mating population will replace the main population's chromosomes whose fitness values are the worst.

In the evaluation stage, the chromosome with the best fitness value is picked from the main population and stored.

In the algorithm, GA will start with initialization, and then continuously repeat the subsequent processes (selection, crossover, mutation, evaluation) until termination condition are reached. The summary of GA algorithm is as follows (Algorithm 2):

Algorithm: Genetic Algorithm

```
Initialization;
Chromosome bestchromosome = chromosome with highest fitness value in main population;

While(termination condition is not reached*)
        Selection; //select the chromosomes to be placed in mating population
        Crossover; //swap chromosome variables in the mating population
        Mutation; //reconstruct chromosome's variables in the mating population
        Replace; //Replace some of main population's chromosomes with mating population's
        Chromosome tempBest = Evaluation; //Obtain the chromosome with highest fitness
        If (tempBest fitness > bestchromosome fitness)
                bestchromosome = tempBest;
        end
end
```

Algorithm 2. Genetic Algorithm

*termination condition used is Number of Generations (check Table 3 in the next section for the value set in the experiment)

Since GA is quite reliant on the generated random chromosomes in the initialization phase, the main population size should be set at high amount. Another alternative is to run the whole GA algorithm multiple times and in each run, the chromosome with best fitness is updated. In the experiment, the second approach is chosen. (Check Table 3 in the next section for the Number of GA Iterations set in the experiment)

2.5.4. GA Parameters

The following table shows the GA parameters used in the experiment (Table 3):

Parameter Name	Parameter Value
Crossover Rate	0.5
Mutation Rate	0.5
Main Population Size	30
Mating Population Size	20
Number of Generations	10
Number of GA Iterations	15

Table 3. GA Parameters used in the experiment

2.6. Split Algorithm

The split algorithm attempts to recursively split a cluster into 2 smaller clusters, based on the change in split fitness values.

2.6.1. Split Fitness

The splitting fitness S is calculated as:

For each attribute $j \in D$ (D *is the* set containing all attributes, i represents each instance in the cluster):

$$d_j = \sum_{i \in CLUSTER}(-|dacc_{ij}|) \quad \textbf{(Eq 5)}$$

$$S = \sum(Top\ half\ highest\ values\ of\ d_j) \quad \textbf{(Eq 6)}$$

dacc is the distance between the instance i and the cluster's attribute mean. In other words, the splitting fitness is the sum of the top 50 percent attributes that are closest to the cluster centroid.

2.6.2. Split Operation

The idea of splitting is to find one attribute which has the highest normalized deviation value, then using that attribute's mean, the cluster is divided into 2 smaller ones. The normalized deviation value is calculated as follows:

For each attribute $j \in D$ (D *is the* set containing all attributes, i represents each instance in the cluster):

$$e_j = \sum_{i \in CLUSTER}\left(\frac{|dacc_{ij}|}{max_j}\right) \quad \textbf{(Eq 7)}$$

dacc is the distance between the instance i and the cluster's attribute mean. max_j is the maximum attribute *j* value in the cluster. The attribute with highest normalized deviation is the attribute with highest *e* value.

After the attribute is selected, each instance in the cluster is then split into 2 clusters; the first cluster contains all instances whose attribute value is higher than the attribute mean, and the second cluster contains the remaining instances. The algorithm will then try to adjust the each of the centroid's attribute value by shifting all the instances among the 2 clusters until the centroids' values no longer change. This part of splitting algorithm is similar to the k-means method.

After splitting, the splitting fitness *S* (Eq 6) of each of the 2 clusters is calculated. If the average of the 2 values is higher than the *S* of original cluster, then the split is considered successful, otherwise the algorithm will roll back to its pre-split state. The algorithm will keep on repeating the splitting phase until splitting fails for all clusters. The summary of split algorithm is as follows (Algorithm 3):

Algorithm: Split

```
Cluster[] c; //cluster array
idx = 0; //index of cluster to be split

do
    for i = 1 to number of attributes
        e[i] = calculate total deviation to cluster centre; //Eq 7
    end

    S = splitting fitness of c[idx]//Eq 6

    int x = index of attribute with highest e;

    Divide(c[idx], x); //split the cluster into 2 smaller clusters based on attribute x
    Cluster c1; Cluster c2;//the first cluster and the second cluster produced
    AverageS = (split fitness of c1 + split fitness of c2)/2; //average split fitness of c1 and c2
    If (AverageS > S)
        Remove c[idx];
        Append c1 and c2 to c[idx] and c[idx+1];
        idx = 0;
    else
        idx++;
        rollback;
    end
while(idx < number of clusters);
```

Algorithm 3. Split Algorithm

2.7. Fusion Algorithm

The fusion algorithm utilizes the sorted attributes found in (Algorithm 1). The fusion algorithm iteratively selects the smallest-sized cluster below the threshold size and places each of the cluster's instances into the closest of remaining clusters. The threshold currently uses the heuristic equation, derived from modified 'rule of thumb' [17]:

$$threshold = N/\sqrt{N/5} \quad \textbf{(Eq 8)}$$

N is the number of instances in the dataset. Basically, the fusion algorithm merges clusters, starting with the smallest sized cluster, until all clusters' sizes are above threshold. The heuristic equation is likely to be removed in the future and replaced with a more stable equation.

Distance calculation to find the closest cluster for an instance will be:

For each attribute $j \in D$ (D is the set of all attributes) and $i \in C$ (C is the set containing all clusters):

$$dist_{ij} = \left|instance_j - centroid_{ij}\right| \quad \textbf{(Eq 9)}$$

Distance to cluster i is then calculated as:

$$distance_i = \sum(Top\ 50\%\ attributes\ with\ highest\ temp\ values\ in\ Algorithm\ 1) \quad \textbf{(Eq 10)}$$

The cluster with the smallest distance (Eq 10) will be selected and the instance will be added to the cluster.

The fusion algorithm can be summarized as follows (Algorithm 4):

Algorithm: Fusion

```
Vector dist; //to store distance values for all clusters

while(there is cluster with size below threshold) //Eq 8
        Cluster c = cluster with smallest size;
        for each instance i in Cluster c
                for each cluster d != c
                        Distance = distance from i to d; //Eq10
                        Append distance to dist Vector;
                end
                Cluster e = cluster with lowest distance value in dist Vector;
                Place i in Cluster e;
        end
        remove Cluster c;
end
```

Algorithm 4. Fusion Algorithm

2.8. After GA, Split, and Fusion

Instances which have been clustered using the above processes will then be removed from the dataset. The whole process is then repeated using the remaining instances in the dataset; the algorithm is completed when dataset becomes empty.

2.9. Class Diagram

The figure below shows the classes, along with the variables and methods, which are used in the algorithm (Fig 4):

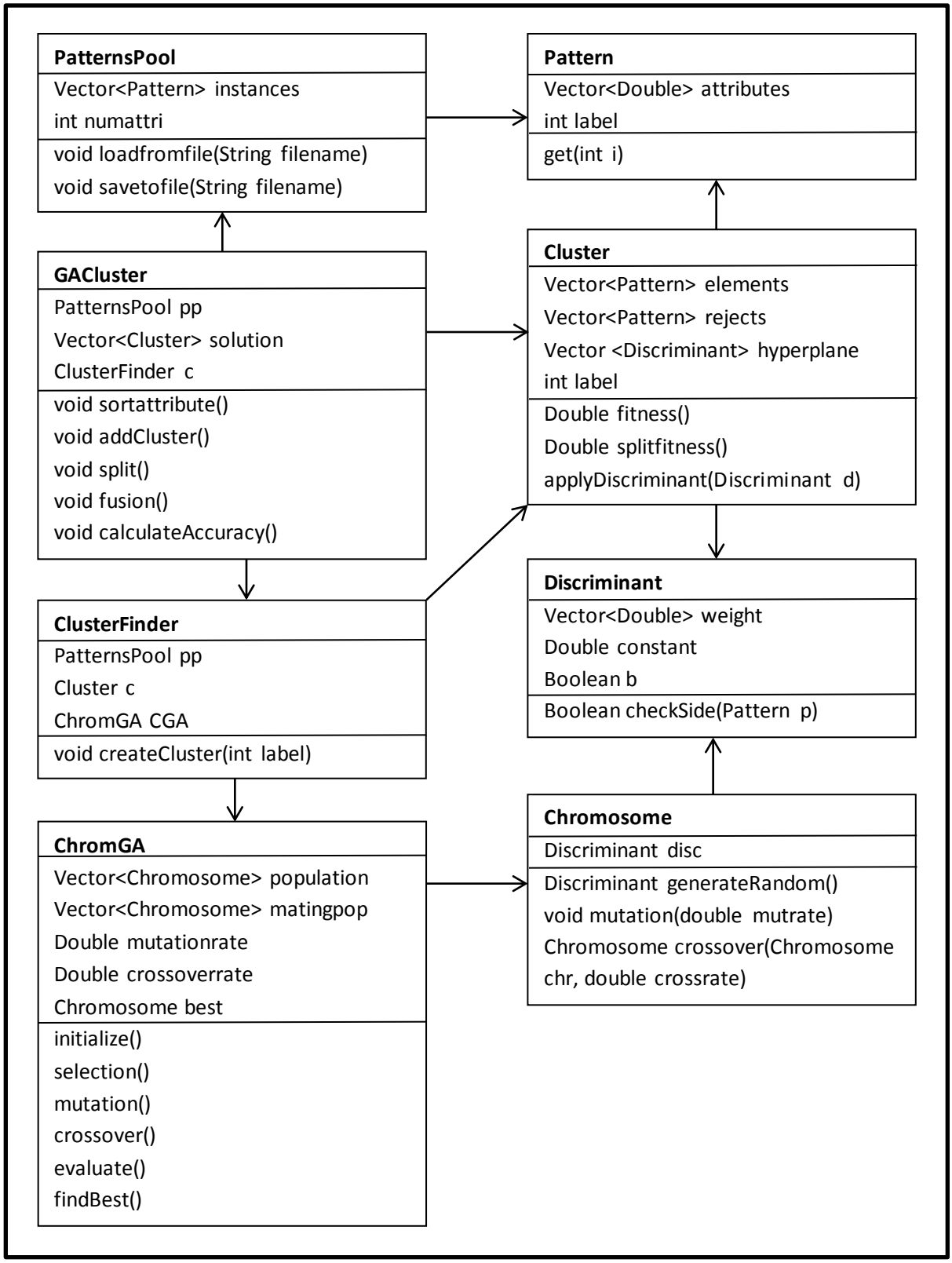

Fig 4. Class Diagram of *GACluster* algorithm

3. Evaluation Design

The evaluation of the *GACluster* algorithm will be based on its clustering accuracy, which is calculated as:

$$accuracy = \frac{total\ dataset\ size - number\ of\ misclustered\ instances}{total\ dataset\ size} * 100\% \qquad \textbf{(Eq 11)}$$

In a cluster, an instance is considered mis-clustered if its actual cluster label is different from the actual cluster label of other instances (Check Table 1 of Input Section to see how the actual cluster labels are represented). If a cluster contains instances of several cluster labels, the label which has the most instances will be the main cluster label. For example, if a cluster of size 100 has 40 instances of label 1, 35 instances of label 2, and 25 instances of label 3, then there will be 60 (35 + 25) mis-clustered instances in the cluster since label 1 dominates the cluster. In the case where there are 2 clusters with the same dominating labels (As an example, the first cluster has 40 instances of label 2 and second cluster has 30 instances of label 2), only the largest number of instances will be considered a valid cluster, the rest of label 2 instances will be classified as mis-clustered.

The clustering accuracy will be done on several datasets from UCI Machine Learning Repository [11]. For each dataset, the algorithm will be run 30 times and the average clustering accuracy will be recorded. The selected datasets are (Table 4):

Dataset Name	Number of Clusters	Number of Instances	Number of Attributes
Iris	3	150	4
Cancer	2	349	9
Wine	3	178	13
Glass	7	214	9

Table 4. UCI Datasets used for Evaluation

In addition to the datasets in Table 4, the testing will also be done on several artificially generated datasets; this is to ensure the flexibility and robustness of *GACluster* algorithm.

The clustering accuracy on UCI datasets will also be compared with other state-of-art clustering algorithms, such as recursive SOM [12] and Information Theoretic Hierarchical Clustering [13]. If *GACluster* manages to produce average clustering accuracy that is higher or close (about 5% accuracy difference) compared to those algorithms for each dataset, then the experiment is considered successful. On the other hand, for the artificial datasets, clustering accuracy of 85% will be considered satisfactory.

Previous experiments had shown that *GACluster* is very time-efficient. For all the UCI datasets, the time required to run the algorithm is always 1 second or less (Windows 7, Core i5 processor); because of this, time-efficiency will not be an issue when evaluating *GACluster*'s performance.

4. Project Plan Review

Since most of the programming work has been completed, subsequent efforts will be focused mainly on improving the heuristics equations and testing. The chart below shows the schedule for remaining work (Fig 5, Fig 6):

Name	Begin date	End date
Continuous Testing and Refinement	12/3/12	2/15/13
Final Evaluation	2/18/13	3/7/13
Final Algorithm Refinement	3/12/13	3/22/13
Experiments	3/25/13	4/2/13
Results and Analysis	4/5/13	4/12/13
Final Dissertation	4/17/13	4/17/13

Fig 5. Schedule of Remaining Tasks

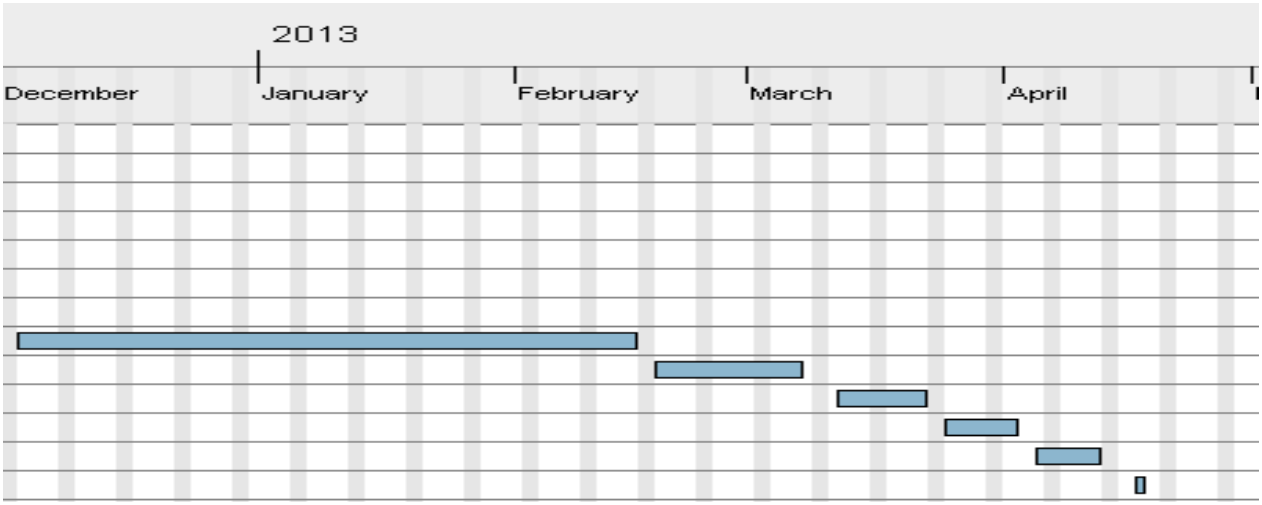

Fig 6. Gantt Chart of Remaining Tasks

References

[1] Han,J. and Kamber,M. (2001) Data Mining: Concepts and Techniques. Morgan Kaufmann, CA, USA.

[2] Maulik, U. and Bandyopadhyay, S. (2000). Genetic algorithm-based clustering technique. Pattern Recognition 33, pp. 1455-1465.

[3] Bhattacharya, A. and De, R. K. (2008). Divisive Correlation Clustering Algorithm (DCCA) for grouping of genes: detecting varying patterns in expression profiles. Bioinformatics 24(11), pp. 1359–1366.

[4] Ankerst M., Breunig M. M., Kriegel, H. P., Sander, J. (1999). "OPTICS: Ordering Points To Identify the Clustering Structure". ACM SIGMOD international conference on Management of data. ACM Press. pp. 49–60.

[5] Parsons, L., Haque, E., Liu, H. (2004). Subspace Clustering for High Dimensional Data: A Review. ACM SIGKDD Explorations Newsletter - Special issue on learning from imbalanced datasets 6(1), pp. 90-105.

[6] Goldberg, D. E. (1989). Genetic Algorithms in Search Optimization and Machine Learning. Addison Wesley. p. 41.

[7] Gallant, S. I. (1990). Perceptron-based learning algorithms. IEEE Transactions on Neural Networks, vol. 1, no. 2, pp. 179–191.

[8] Urbakh, V. Y. (1972). A discriminant method of clustering. Journal of Multivariate Analysis, vol 2, no. 3, pp. 249-260.

[9] Iris Flower Data Set, Wikipedia, available online at: http://en.wikipedia.org/wiki/Iris_flower_data_set, Nov.22nd 2011.

[10] Haykin, S. (1998). Neural Networks: A Comprehensive Foundation (2 ed.). Prentice Hall.

[11] UCI Machine Learning Repository: http://www.ics.uci.edu/~mlearn/MLRepository.html

[12] Ramanathan, K. and Guan, S. U. (2006). Recursive Self Organizing Maps with Hybrid Clustering. IEEE CIS.

[13] Aghagolzadeh M., Soltanian-Zadeh H., Araabi B. N. (2011). Information Theoretic Hierarchical Clustering. Entropy, 13, pp. 450-465.

[14] Statistical Randomness, Wikipedia, available online at: http://en.wikipedia.org/wiki/Statistical_randomness, Nov.22nd 2011.

[15] Tianlin Y., Guan S. U., Final Year Project Design Report, Information and Computing Science, Xi'an Jiaotong-Liverpool University, 2011.

[16] DeJong K. A., "Learning with genetic algorithms: an overview," Mach. Learn., vol.3, pp.121-138, 1988.

[17] Determining the number of clusters in a data set, Wikipedia, available online at: http://en.wikipedia.org/wiki/Determining_the_number_of_clusters_in_a_data_set, Nov.22nd 2011.