

# **IMPLEMENTATION OF K-MEANS AND GENETIC K-MEANS ALGORITHM**

**By**

**ANKIT GARG - 051233**

**AKSHAT JAIN – 051284**

**PRIYANKA GARG – 051285**

**PRATEEK AGARWAL – 051292**

**NIRJHAR VERMANI - 051322**



**MAY-2009**

**Submitted in partial fulfillment of the Degree of Bachelor of  
Technology**

**DEPARTMENT OF COMPUTER SCIENCE ENGINEERING  
JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY-  
WAKNAGHAT**

**CERTIFICATE**

This is to certify that the work entitled, “Data Clustering Using Data Mining Tool” submitted by Ankit Garg-051233, Akshat Jain-051284, Priyanka Garg-051285, Prateek Agarwal-051292 and Nirjhar Virmani-051322 in partial fulfillment for the award of degree of Bachelor of Technology in Computer Science Engineering of Jaypee University of Information Technology has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

[ Satish Chandra]

[ Pardeep Kumar]

## ACKNOWLEDGMENT

We would like to express our deep sense of gratitude to our project guide Mr. Satish Chandra , Department of Computer Science Engineering, who gave us the opportunity to work on this project and co-operated with us in learning about the working of the project and guided us throughout the year

We are highly indebted to Mr. Pardeep Kumar whose help, stimulating suggestions and encouragement helped us in all stages of the project. His overly enthusiasm and his view for providing ‘Only high quality and not less’ has made a deep impression on us.

Last, but not the least, we would like to thank everyone who has contributed for successful completion of the project.

Ankit Garg

Akshat Jain

Priyanka Garg

Prateek Agarwal

Nirjhar Virmani

## TABLE OF CONTENTS

1) Introduction.....	8
1.1 Problem Statement.....	8
1.2 Data Mining.....	8
1.3 Data Clustering.....	9
1.3.1 Clustering Techniques.....	9
1.3.2 Partitioning Algorithms: Basic Concept.....	10
1.3.3 Hierarchical Algorithms.....	10
1.3.4 Density Based Clustering Methods.....	11
1.3.5 Grid Based Clustering Methods.....	11
1.3.6 Model Based Clustering Methods.....	11
1.4 KDD Process.....	12
1.4.1 KDD Process Diagram.....	13
1.5 K-Means Algorithm.....	14
1.6 Genetic Algorithm.....	14
1.7 Genetic K-Means Algorithm.....	16
2) K-means.....	16
2.1 Description.....	16
2.2 Algorithm / Pseudo Code.....	17
2.3 Flowchart For K-Means.....	18
2.3.1 Flow Chart Explanation.....	18
2.4 Example.....	20
2.5 Limitations.....	26
3) Genetic Algorithm.....	27
3.1 Initialization.....	28
3.2 Selection.....	29
3.3 Reproduction.....	29
3.4 Termination.....	30

3.5 Pseudo Code.....	30
3.6 Flowchart.....	31
3.7 Observations.....	32
3.8 Variants.....	34
4) Genetic K-means.....	36
4.1 Why Genetic K-Means Algorithm? .....	36
4.2 Introduction.....	36
4.3 Methodology.....	37
4.4 Flowchart.....	38
5) Code Implementation and Results with Snapshots.....	39
5.1 Centroid Code.....	39
5.2 Cluster Code.....	43
5.3 Data Point Code.....	45
5.4 GA Implementation Code.....	49
5.5 GA Main Code.....	54
5.6 Interface Code.....	54
5.7 JCA Code.....	61
5.8 Prg Main Code.....	73
5.9 Results.....	76
5.9.1 Snapshots with K-Means Selected.....	76
5.9.2 Snapshots with Hybrid Selected.....	79
5.9.3 Snapshots of the Dataset Used.....	83
Conclusion.....	85
Bibliography.....	86
A. Appendix.....	87
i) How To Install JDK?.....	87
ii) How To Install JDBC?.....	91

## LIST OF FIGURES

Fig.1	KDD Process Diagram
Fig 2	various stages of clustering
Fig 3	Flowchart of K-means algorithm
Fig 4	Plot of data points in a given set of samples
Fig 5	Initial cluster centers
Fig 6	Cluster assignment
Fig 7	New cluster centers
Fig 8	Formation of new clusters
Fig 9	Formation of clusters
Fig 10	Final clusters
Fig 11	Flowchart of Genetic algorithm
Fig 12	Flowchart of Genetic K-means algorithm
Fig 13	Snapshot with K-means selected
Fig 14	Result of K-means
Fig 15	x-y coordinates of data points in first cluster
Fig 16	x-y coordinates of data points in second cluster
Fig 17	x-y coordinates of data points in third cluster
Fig 18	Snapshot with hybrid selected
Fig 19	Result of hybrid
Fig 20	x-y coordinates of data points in first cluster
Fig 21	x-y coordinates of data points in second cluster
Fig 22	x-y coordinates of data points in third cluster
Fig 23	snapshot of data set

## LIST OF ABBREVIATIONS

- 1) KMA – K-Means Algorithm
- 2) GA – Genetic Algorithm
- 3) GKA – Genetic K-Means Algorithm
- 4) KDD – Knowledge Data Discovery
- 5) JDK – Java Development Kit
- 6) JDBC – Java Data Base Connectivity
- 7) FWI – Forest Fire Weather Index
- 8) FFMC – Fine Fuel Moisture Code
- 9) DMC – Duff Moisture Code
- 10) DC – Drought Code
- 11) ISI – Initial Spread Index
- 12) BUI – Build Up Index

# CHAPTER-I

## INTRODUCTION

### **1.1 Problem Statement**

Implementation of K-Means, Genetic Algorithm and Hybrid of both K-means and Genetic Algorithm for prediction of forest fires.

### **1.2 Data Mining**

It is a process of extracting data from a data ware house where the data can be non-trivial, implicit, unknown, potentially useful. As more data is gathered, with the amount of data doubling every three years, data mining is becoming an increasingly important tool to transform this data into information.

Humans have been “manually” extracting information from data for centuries, but the increasing volume of data in modern times has called for more automatic approaches. As data sets and the information extracted from them has grown in size and complexity, direct hands-on data analysis has increasingly been supplemented and augmented with indirect, automatic data processing using more complex and sophisticated tools, methods and models. The proliferation, ubiquity and increasing power of computer technology has aided data collection, processing, management and storage. However, the captured data needs to be converted into information and knowledge to become useful. Data mining is the process of using computing power to apply methodologies, including new techniques for knowledge discovery to data.

### **1.3 Data Clustering**

Clustering is the assignment of objects into groups (called *clusters*) so that objects from the same cluster are more similar to each other than objects from different clusters. Often similarity is assessed according to a distance measure. Clustering is a common technique for statistical data analysis, which is used in many fields, including machine learning, data mining, pattern recognition, image analysis and bioinformatics.



Common distance functions:

- The Euclidean distance : A review of cluster analysis in health psychology research found that the most common distance measure in published studies in that research area is the Euclidean distance or the squared Euclidean distance.
- The Manhattan distance
- The maximum norm
- The Mahalanobis distance corrects data for different scales and correlations in the variables
- The angle between two vectors can be used as a distance measure when clustering high dimensional data
- The Hamming distance measures the minimum number of substitutions required to change one member into another.

### ***1.3.1 Clustering Techniques***

- Partitioning algorithms: Construct various partitions and then evaluate them by some criterion
- Hierarchy algorithms: Create a hierarchical decomposition of the set of data (or objects) using some criterion
- Density-based: based on connectivity and density functions
- Grid-based: based on a multiple-level granularity structure
- Model-based: A model is hypothesized for each of the clusters and the idea is to find the best fit of that model to each other

### ***1.3.2 Partitioning Algorithms: Basic Concept***

- Partitioning method: Construct a partition of a database  $D$  of  $n$  objects into a set of  $k$  clusters
- Given a  $k$ , find a partition of  $k$  clusters that optimizes the chosen partitioning criterion

- Global optimal: exhaustively enumerate all partitions
- Heuristic methods: *k-means* and *k-medoids* algorithms
- *k-means* (MacQueen'67): Each cluster is represented by the center of the cluster
- *k-medoids* or PAM (Partition around medoids) (Kaufman & Rousseeuw'87): Each cluster is represented by one of the objects in the cluster
- Classical partitioning:
  - K-Means (Centroid-based technique)
  - K-Medoids (Representative point based technique)
  - K-Modes (Huang 1998)
- Partitioning methods in large databases
  - CLARA (Clustering large Applications)
  - CLARANS (Clustering Large Applications based upon Randomized Search)

### ***1.3.3 Hierarchical Algorithms:***

- It can be classified into either agglomerative or divisive
- Agglomerative Approach (bottom-up): set each object as a individual cluster or group and merges the objects or groups close to one another, until all of the groups are merged into one(the topmost level of the hierarchy)
- Divisive Approach (top down): starts with all objects in the same cluster. In each successive iteration, a cluster is split up into smaller clusters, until a termination condition holds.
- Integration of hierarchical with distance-based clustering
  - BIRCH (1996) : uses CF-tree and incrementally adjusts the quality of sub-clusters

- CURE (1998) : selects well-scattered points from the cluster and then shrinks them towards the center of the cluster by a specified fraction
- CHAMELEON (1999) : hierarchical clustering using dynamic modeling

#### ***1.3.4 Density-Based Clustering Methods***

- DBSCAN : Ester, et al. (KDD'96)
- OPTICS : Ankerst, et al (SIGMOD'99).
- DENCLUE : Hinneburg & D. Keim (KDD'98)
- CLIQUE : Agrawal, et al. (SIGMOD'98)

#### ***1.3.5 Grid-Based Clustering Method***

- Several interesting methods
  - STING (a Statistical Information Grid approach) by Wang, Yang and Muntz (1997)
  - WaveCluster by Sheikholeslami, Chatterjee, and Zhang (VLDB'98)
    - A multi-resolution clustering approach using wavelet method
  - CLIQUE: Agrawal, et al. (SIGMOD'98)

#### ***1.3.6 Model-Based Clustering Methods***

- COBWEB (Fisher'87)
  - A popular a simple method of incremental conceptual learning
  - Creates a hierarchical clustering in the form of a classification tree
  - Each node refers to a concept and contains a probabilistic description of that concepts

## **1.4 KDD Process**

Knowledge discovery in databases (KDD) is the non-trivial extraction of implicit, previously unknown, and potentially useful information from databases.

Six common and essential elements qualify each as a knowledge discovery technique. The following are basic features that all KDD techniques share:

- All approaches deal with large amounts of data
- Efficiency is required due to volume of data
- Accuracy is an essential element
- All require the use of a high-level language
- All approaches use some form of automated learning
- All produce some interesting results

Large amounts of data are required to provide sufficient information to derive additional knowledge. Since large amounts of data are required, processing efficiency is essential. Accuracy is required to assure that discovered knowledge is valid. The results should be presented in a manner that is understandable by humans. One of the major premises of KDD is that the knowledge is discovered using intelligent learning techniques that sift through the data in an automated process. For this technique to be considered useful in terms of knowledge discovery, the discovered knowledge must be interesting; that is, it must have potential value to the user.

KDD provides the capability to discover new and meaningful information by using existing data. KDD quickly exceeds the human capacity to analyze large data sets. The amount of data that requires processing and analysis in a large database exceeds human capabilities, and the difficulty of accurately transforming raw data into knowledge surpasses the limits of traditional databases. Therefore, the full utilization of stored data depends on the use of knowledge discovery techniques.

The usefulness of future applications of KDD is far-reaching. KDD may be used as a means of information retrieval, in the same manner that intelligent agents perform information retrieval on

the web. New patterns or trends in data may be discovered using these techniques. KDD may also be used as a basis for the intelligent interfaces of tomorrow, by adding a knowledge discovery component to a database engine or by integrating KDD with spreadsheets and visualizations.

#### 1.4.1 KDD Process Diagram

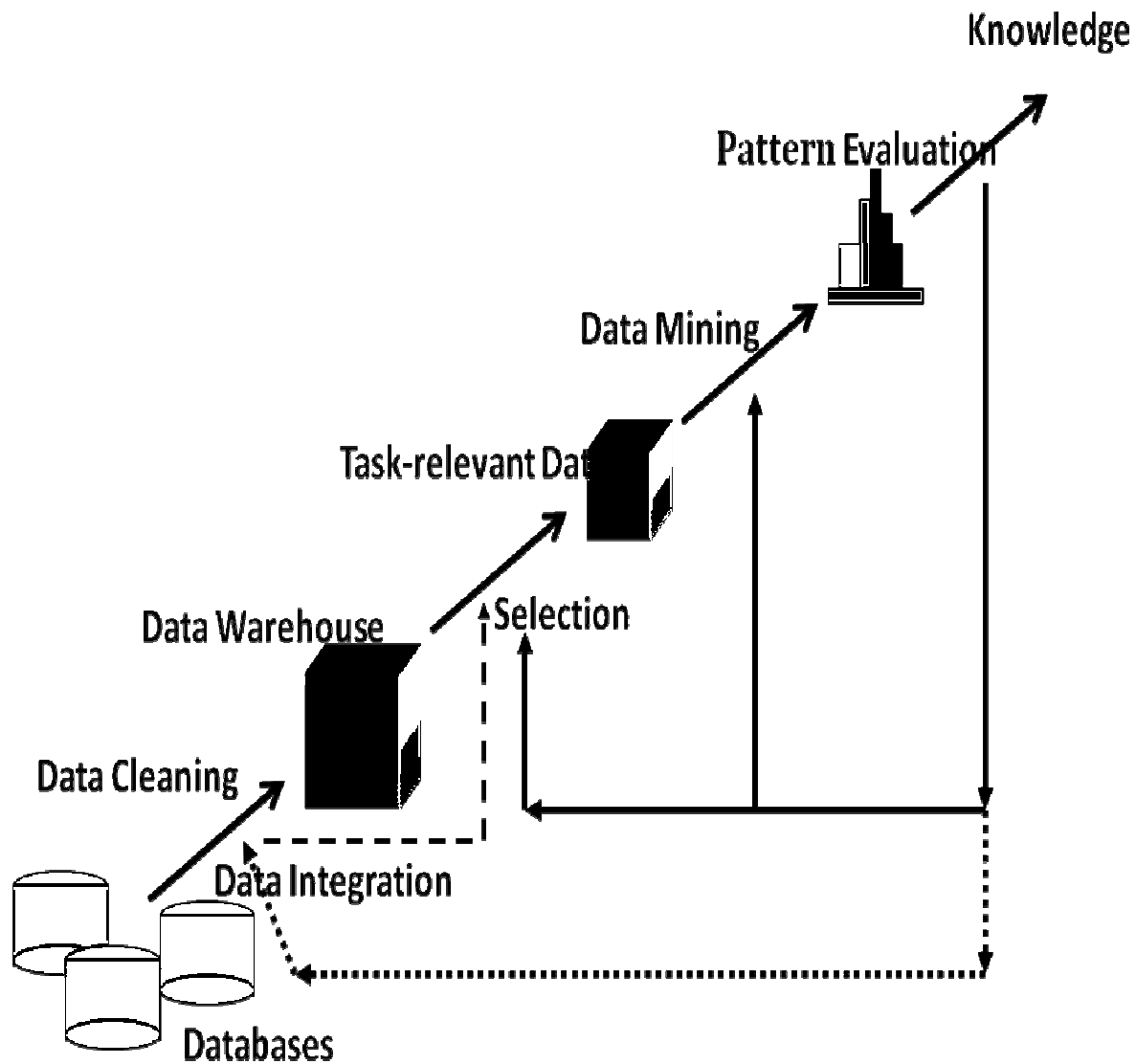


FIG. 1

### **1.5 K-Means Algorithm**

K-means (MacQueen, 1967) is one of the simplest unsupervised learning algorithms that solve the well known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed a priori. The main idea is to define k centroids, one for each cluster. These centroids should be placed in a cunning way because of different location causes different result. So, the better choice is to place them as much as possible far away from each other. The next step is to take each point belonging to a given data set and associate it to the nearest centroid. When no point is pending, the first step is completed and an early groupage is done. At this point we need to re-calculate k new centroids of the clusters resulting from the previous step. After we have these k new centroids, a new binding has to be done between the same data set points and the nearest new centroid. A loop has been generated. As a result of this loop we may notice that the k centroids change their location step by step until no more changes are done. In other words centroids do not move any more.

### **1.6 Genetic Algorithm**

Genetic algorithms were formally introduced in the United States in the 1970s by John Holland at University of Michigan. A **genetic algorithm (GA)** is a search technique used in computing to find exact or approximate solutions to optimization and search problems.

Genetic algorithms are a particular class of evolutionary algorithms that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover (also called recombination)

Genetic algorithms operate on set of possible solutions. Because of random nature of the genetic algorithm, solutions found by the algorithm can be good, poor or infeasible [defective, erroneous] so there should be a way to specify how good that solution is. This is done by assigning fitness value [or just fitness] to the solution. Chromosomes represent solutions within the genetic algorithm. Two basic component of chromosome are coded solution and its fitness value.

Chromosomes are grouped into population [set of solutions] on which the genetic algorithm operates. In each step [generation] genetic algorithm selects chromosomes from population [selection is usually based on fitness value of chromosome] and combines them to produce new chromosomes [offspring]. These offspring chromosomes form new population [or replace some of the chromosomes in the existing population] in hope that new population will be better than previous. Populations keep track of the worst and the best chromosomes and store additional statistical information which can be used by genetic algorithm to determine stop criteria.

Genetic algorithms produce new chromosomes [solutions] by combining existing chromosomes. This operation is called crossover. Crossover operation takes parts of solution encodings from two existing chromosomes [parents] and combines them into single solution [new chromosome]. This operation depends on chromosome representation and can be very complicated. Although general crossover operations are easy to implement, building specialized crossover operation for specific problem can greatly improve performance of the genetic algorithm

### **1.7 Genetic K-Means Algorithm**

A Hybrid Genetic Algorithm (GA) that finds a globally optimal partition of a given data into a specified number of clusters. GA's used earlier in clustering employ either an expensive crossover operator to generate valid child chromosomes from parent chromosomes or a costly fitness function or both. To circumvent these expensive operations, we hybridize GA with a classical gradient descent algorithm used in clustering, K-means algorithm. Hence, the name Genetic K-means Algorithm (GKA). We define K-means operator, one-step of K-means algorithm, and use it in GKA as a search operator instead of crossover. It is observed in the simulations that GKA converges to the best known optimum corresponding to the given data in concurrence with the convergence result.

## CHAPTER – 2

### K-Means Algorithm

In statistics and machine learning, **k-means clustering** is a method of cluster analysis which aims to partition  $n$  observations into  $k$  clusters in which each observation belongs to the cluster with the nearest mean. In a given database of  $N$  objects and where  $K$  is the number of clusters to form such that the objects within a cluster are similar on the basis of distance. It is a centroid based technique. Clustering is done in such a manner that intra-cluster similarity is high and inter-cluster similarity is low

#### 2.1 Description

Given a set of observations  $(x_1, x_2, \dots, x_n)$ , where each observation is a  $d$ -dimensional real vector, then  $k$ -means clustering aims to partition this set into  $k$  partitions ( $k < n$ )  $S = \{S_1, S_2, \dots, S_k\}$  so as to minimize the within-cluster sum of squares (WCSS):

$$\arg \min_{\mathbf{S}} \sum_{i=1}^k \sum_{x_j \in S_i} \|x_j - \mu_i\|^2$$

Where  $\mu_i$  is the mean of  $S_i$ .

Also, the distance function used in our software is the Euclidian Distance given by the formula

$$d(X, Y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}.$$

The function for the initial cluster centers to be arbitrarily chosen is

$$\text{Initial Cluster Centers} = \text{Max value} - \text{min value} / ((\text{cluster length} + 1) * n + \text{min value})$$

After, the initial cluster centers are obtained each data point is assigned to a cluster to which the data point is closest based on the Euclidian Distance and thus initial clusters are formed. Now, iteratively new cluster centers are formed based on the mean of all data points in a given cluster



and again classified into new clusters by the closest Euclidian Distance. This process continues till final clusters are formed

## **2.2 Psuedo Code For K-Means Algorithm**

**Algorithm** . (*k*-means) The *k*-means algorithm for partitioning based on the mean value of the objects in the cluster.

**Input:** The number of clusters *k*, and a database containing *n* objects.

**Output:** A set of *k* clusters which minimizes the squared-error criterion.

**Method:** The *k*-means algorithm is implemented as follows.

- 1) arbitrarily choose *k* objects as the initial cluster centers;
- 2) repeat
- 3)     (re)assign each object to the cluster to which the object is the most similar,  
       based on the mean value of the objects in the cluster;
- 4)     update the cluster means, i.e., calculate the mean value of the objects for each cluster;
- 5) until no change;

□

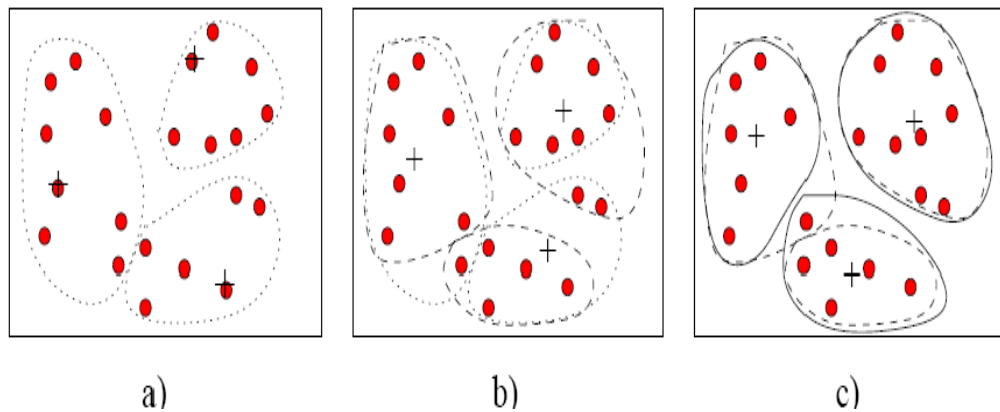


Fig 2

### 2.3 Flow Chart For K-means Algorithm

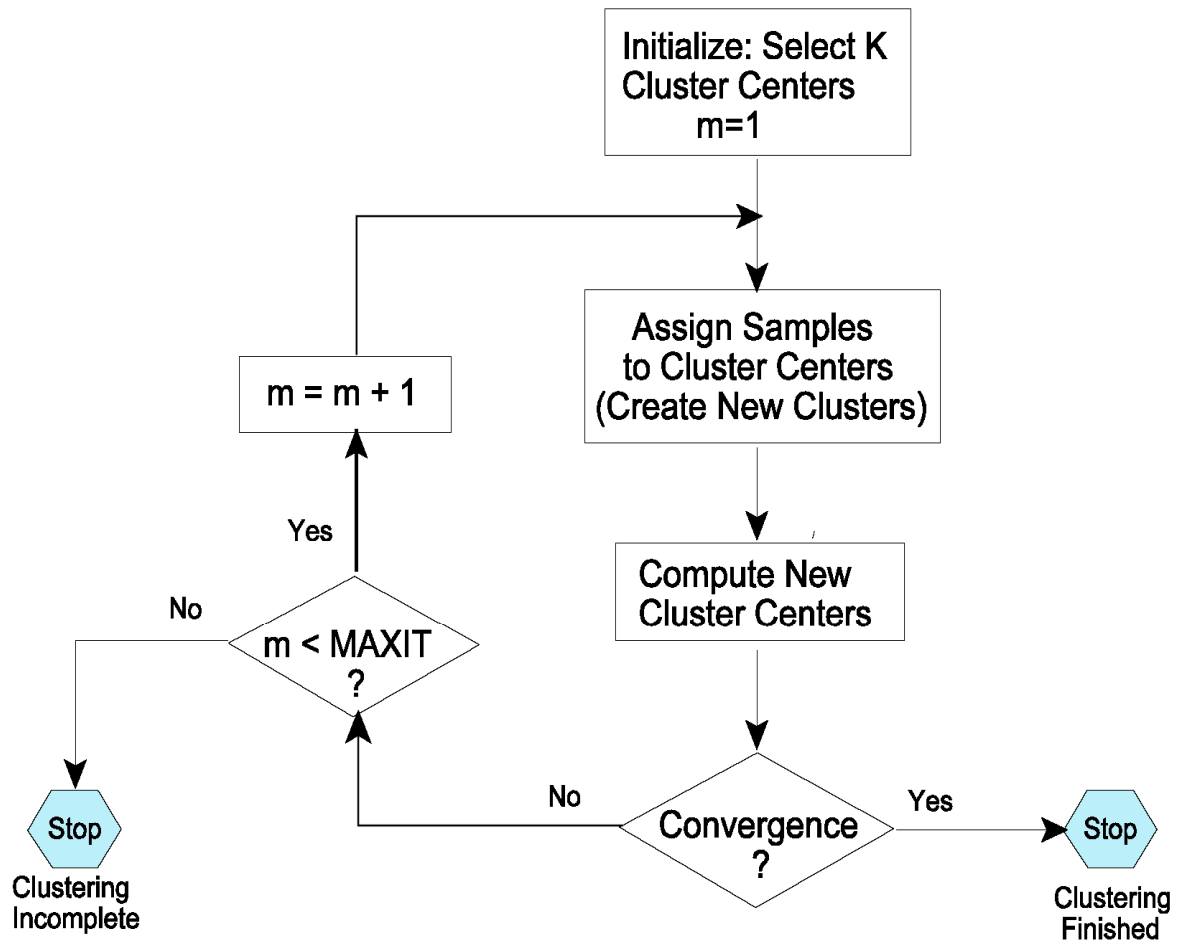


Fig 3

#### 2.3.1 Flow Chart Explanation

##### Step 1 Initialization

Choose K initial Cluster centers

$$M_1(1), M_2(1), \dots, M_K(1)$$

Method 1 – First K samples

Method 2 – K data samples selected randomly

Method 3 – K random vectors

Set  $m = 1$  and Go To Step 2

*Step 2 Determine New Clusters*

Using Cluster centers Distribute pattern vectors using minimum distance.

Method 1 – Use Euclidean distance

Method 2 – Use other distance measures

Assign sample  $x_j$  to class  $C_k$  if

$$\|x_j - M_k(m)\| < \|x_j - M_i(m)\|$$

for all  $i=1, 2, \dots, K$  and  $i \neq k$

Go to Step 3

*Step 3 Compute New Cluster Centers*

Using the new Cluster assignment

$$Cl_k(m) \quad m = 1, 2, \dots, K$$

Compute new cluster centers

$$M_k(m+1) \quad m = 1, 2, \dots, K$$

Using :-

$$M_k(m+1) = \frac{1}{N_k} \sum_{x_j \in Cl_k(m)} x_j$$

where  $N_k$ ,  $k = 1, 2, \dots, K$  is the number of pattern vectors in  $Cl_k(m)$

Go to Step 4

*Step 4 Checks for Convergence*

Using Cluster centers from step 3 check for convergence

Convergence occurs if the means do not change

If  $M_k(m+1) = M_k(m)$   
for  $k=1, 2, \dots, K$

If Convergence occurs Clustering is complete and the results given.

If No Convergence then Go to Step 5

#### Step 5 Checks for Maximum Number of Iterations

Define MAXIT as the maximum number of iterations that is acceptable.

If  $m = \text{MAXIT}$  Then display no convergence And Stop.

If  $m < \text{MAXIT}$  Then  $m=m+1$  (increment  $m$ )

And Return to Step 2

### **2.4 Example**

Given the following set of pattern vectors

$$\mathbf{x}_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad \mathbf{x}_2 = \begin{pmatrix} 1 \\ 3 \end{pmatrix}, \quad \mathbf{x}_3 = \begin{pmatrix} 2 \\ 2 \end{pmatrix},$$

$$\mathbf{x}_4 = \begin{pmatrix} 4 \\ 4 \end{pmatrix}, \quad \mathbf{x}_5 = \begin{pmatrix} 4 \\ 5 \end{pmatrix}, \quad \mathbf{x}_6 = \begin{pmatrix} 5 \\ 4 \end{pmatrix},$$

$$\mathbf{x}_7 = \begin{pmatrix} 5 \\ 5 \end{pmatrix}$$

Plot of Data points in Given set of samples

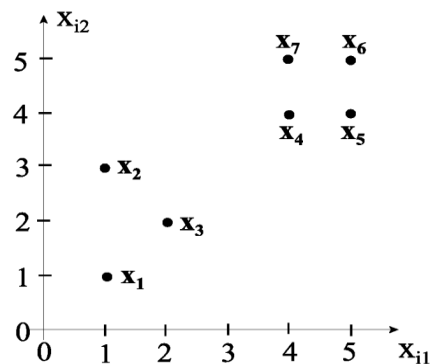


Fig 4

- (a) Using the first two samples  $X_1$  and  $X_2$ , as initial cluster center for the K-Means algorithm find a separation of the pattern vector into two clusters.
- (b) Using the first three samples,  $X_1$ ,  $X_2$ , and  $X_3$  as initial cluster centers for the K-Means algorithm find a separation of the pattern vectors into three clusters.

(a) Solution – 2-class case

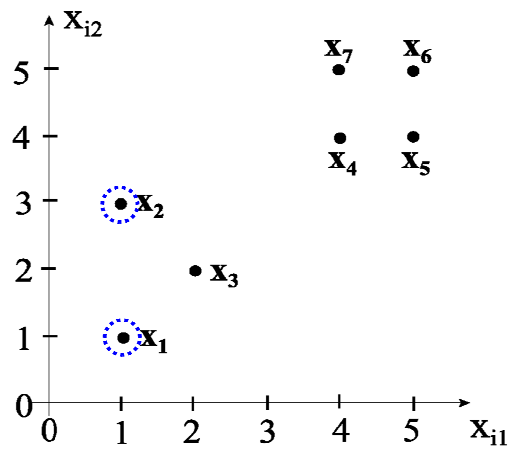


Fig 5

Initial Cluster Centers

$$M_1(1) = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad M_2(1) = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

Distances from all Samples to cluster centers

---	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$
$d(x_i, M_1(1))$	0	2	1.4142	4.2426	5	5	5.6569
$d(x_i, M_2(1))$	2	0	1.4142	3.1623	3.6055	4.1231	4.4721
Assignment	$Cl_1$	$Cl_2$	$Cl_1$	$Cl_2$	$Cl_2$	$Cl_2$	$Cl_2$

First Cluster assignment

$$Cl_1(1) = \{X_1, X_3\}$$

$$Cl_2(1) = \{x_2, x_4, x_5, x_6, x_7\}$$

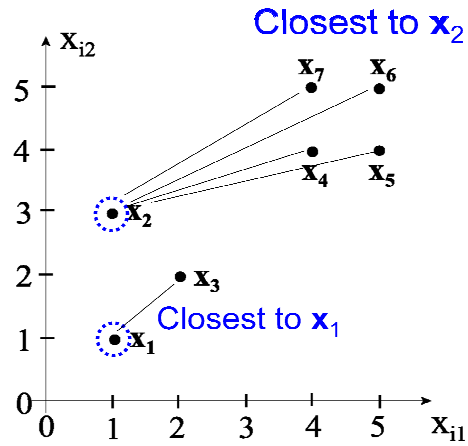


Fig 6

First Cluster Assignment

$$Cl_1(1) = \{x_1, x_3\}$$

$$Cl_2(1) = \{x_2, x_4, x_5, x_6, x_7\}$$

Compute New Cluster centers

$$\begin{aligned} M_1(2) &= \frac{1}{2} (x_1 + x_3) = \frac{1}{2} \left( \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 2 \\ 2 \end{bmatrix} \right) = \begin{bmatrix} 1.5 \\ 1.5 \end{bmatrix} \\ M_2(2) &= \frac{1}{5} (x_2 + x_4 + x_5 + x_6 + x_7) \\ &= \frac{1}{5} \left( \begin{bmatrix} 1 \\ 3 \end{bmatrix} + \begin{bmatrix} 4 \\ 4 \end{bmatrix} + \begin{bmatrix} 4 \\ 5 \end{bmatrix} + \begin{bmatrix} 5 \\ 4 \end{bmatrix} + \begin{bmatrix} 5 \\ 5 \end{bmatrix} \right) = \begin{bmatrix} 3.8 \\ 4.2 \end{bmatrix} \end{aligned}$$

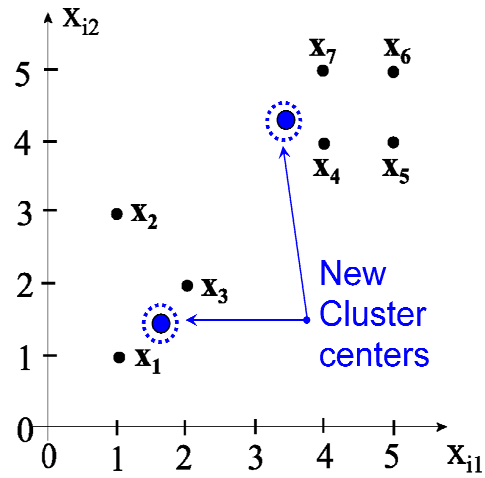


Fig 7

Distances from all Samples to cluster centers

---	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$
$d(x_i, \mathbf{M}_1(1))$	0.7071	1.5811	0.7071	3.5355	4.3012	4.3012	4.9497
$d(x_i, \mathbf{M}_2(1))$	4.2521	3.0463	2.8425	0.2828	0.8246	1.2166	1.4422
Assignment	$Cl_1$	$Cl_1$	$Cl_1$	$Cl_2$	$Cl_2$	$Cl_2$	$Cl_2$

Second Cluster assignment

$$Cl_1(2) = \{x_1, x_2, x_3\}$$

$$Cl_2(2) = \{x_4, x_5, x_6, x_7\}$$

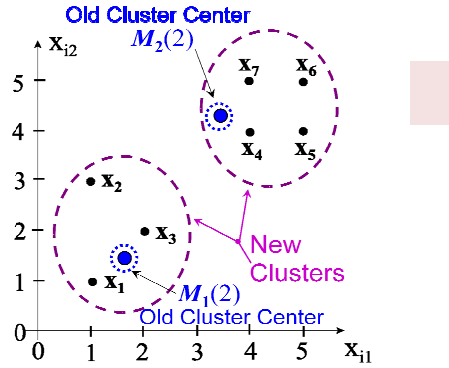


Fig 8

Compute New Cluster Centers

$$\begin{aligned} M_1(2) &= \frac{1}{3} (x_1 + x_2 + x_3) \\ &= \frac{1}{2} \left( \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 2 \\ 2 \end{bmatrix} \right) = \begin{bmatrix} 1.33 \\ 2 \end{bmatrix} \end{aligned}$$

$$\begin{aligned} M_2(2) &= \frac{1}{4} (x_4 + x_5 + x_6 + x_7) \\ &= \frac{1}{4} \left( \begin{bmatrix} 4 \\ 4 \end{bmatrix} + \begin{bmatrix} 4 \\ 5 \end{bmatrix} + \begin{bmatrix} 5 \\ 4 \end{bmatrix} + \begin{bmatrix} 5 \\ 5 \end{bmatrix} \right) = \begin{bmatrix} 4.5 \\ 4.5 \end{bmatrix} \end{aligned}$$

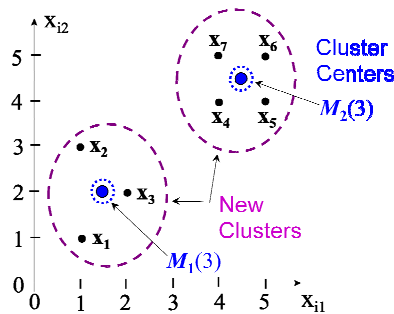


Fig 9

Distances from all Samples to cluster centers

---	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$
$d(x_i, M_1(3))$	1.0541	1.0541	0.6667	3.3333	4.0138	4.1767	4.7376
$d(x_i, M_2(3))$	4.9497	3.8079	3.5355	0.7071	0.7071	0.7071	0.7071
Assignment	$Cl_1$	$Cl_1$	$Cl_1$	$Cl_2$	$Cl_2$	$Cl_2$	$Cl_2$



Compute New Cluster centers

$$Cl_1(3) = Cl_1(2) \quad Cl_2(3) = Cl_2(2)$$

$$M_1(3) = M_1(2) \quad M_2(3) = M_2(2)$$

$$Cl_1(final) = \{ \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3 \}$$

$$Cl_2(final) = \{ \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7 \}$$

(b) Solution: 3-Class case

Select Initial Cluster Centers

$$M_1(1) = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad M_2(1) = \begin{bmatrix} 1 \\ 3 \end{bmatrix} \quad M_3(1) = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

First Cluster assignment using distances from pattern vectors to initial cluster centers

$$Cl_1(1) = \{ \mathbf{x}_1 \} \quad Cl_2(1) = \{ \mathbf{x}_2, \mathbf{x}_5 \}$$

$$Cl_3(1) = \{ \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_6, \mathbf{x}_7 \}$$

Compute New Cluster centers

$$M_1(3) = \begin{bmatrix} 1.5 \\ 1.5 \end{bmatrix} \quad M_2(3) = \begin{bmatrix} 1 \\ 3 \end{bmatrix} \quad M_3(3) = \begin{bmatrix} 4.5 \\ 4.5 \end{bmatrix}$$

Second Cluster assignment using distances from pattern vectors to cluster centers

$$Cl_1(2) = \{ \mathbf{x}_1, \mathbf{x}_3 \} \quad Cl_2(2) = \{ \mathbf{x}_2 \}$$

$$Cl_3(2) = \{x_4, x_5, x_6, x_7\}$$

At the next step we have convergence as the cluster centers do not change thus the Final Cluster Assignment becomes

$$Cl_1(final) = \{x_1, x_3\}$$

$$Cl_2(final) = \{x_2\}$$

$$Cl_3(final) = \{x_4, x_5, x_6, x_7\}$$

Final 3-Class Clusters

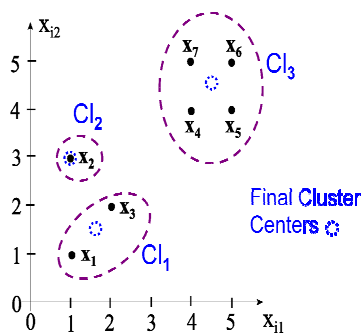


Fig 10

## 2.5 Limitations

- 1) K-Means algorithm has problems when clusters are of differing Sizes, Densities and Non-Globular Size.
- 2) K-Means algorithm has problems when the data contains Outliers.

## CHAPTER – 3

### GENETIC ALGORITHM

A **genetic algorithm (GA)** is a search technique used in computing to find exact or approximate solutions to optimization and search problems. Genetic algorithms are categorized as global search heuristics. Genetic algorithms are a particular class of evolutionary algorithms (also known as evolutionary computation) that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover (also called recombination).

Genetic algorithms are implemented in a computer simulation in which a population of abstract representations (called chromosomes or the genotype of the genome) of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem evolves toward better solutions. Traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible. The evolution usually starts from a population of randomly generated individuals and happens in generations. In each generation, the fitness of every individual in the population is evaluated, multiple individuals are stochastically selected from the current population (based on their fitness), and modified (recombined and possibly randomly mutated) to form a new population. The new population is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. If the algorithm has terminated due to a maximum number of generations, a satisfactory solution may or may not have been reached.

Genetic algorithms find application in bioinformatics, phylogenetics, computational science, engineering, economics, chemistry, manufacturing, mathematics, physics and other fields.

A typical genetic algorithm requires:

1. a genetic representation of the solution domain,
2. a fitness function to evaluate the solution domain.

A standard representation of the solution is as an array of bits. Arrays of other types and structures can be used in essentially the same way. The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size, which facilitates simple crossover operations. Variable length representations may also be used, but crossover implementation is more complex in this case. Tree-like representations are explored in genetic programming and graph-form representations are explored in evolutionary programming.

The fitness function is defined over the genetic representation and measures the *quality* of the represented solution. The fitness function is always problem dependent. For instance, in the knapsack problem one wants to maximize the total value of objects that can be put in a knapsack of some fixed capacity. A representation of a solution might be an array of bits, where each bit represents a different object, and the value of the bit (0 or 1) represents whether or not the object is in the knapsack. Not every such representation is valid, as the size of objects may exceed the capacity of the knapsack. The *fitness* of the solution is the sum of values of all objects in the knapsack if the representation is valid, or 0 otherwise. In some problems, it is hard or even impossible to define the fitness expression; in these cases, interactive genetic algorithms are used.

Once we have the genetic representation and the fitness function defined, GA proceeds to initialize a population of solutions randomly, then improve it through repetitive application of mutation, crossover, inversion and selection operators.

### **3.1 Initialization**

Initially many individual solutions are randomly generated to form an initial population. The population size depends on the nature of the problem, but typically contains several hundreds or thousands of possible solutions. Traditionally, the population is generated randomly, covering the entire range of possible solutions (the *search space*). Occasionally, the solutions may be "seeded" in areas where optimal solutions are likely to be found.

### **3.2 Selection**

During each successive generation, a proportion of the existing population is selected to breed a new generation. Individual solutions are selected through a *fitness-based* process, where fitter solutions (as measured by a fitness function) are typically more likely to be selected. Certain selection methods rate the fitness of each solution and preferentially select the best solutions. Other methods rate only a random sample of the population, as this process may be very time-consuming.

Most functions are stochastic and designed so that a small proportion of less fit solutions are selected. This helps keep the diversity of the population large, preventing premature convergence on poor solutions. Popular and well-studied selection methods include roulette wheel selection and tournament selection.

### **3.3 Reproduction**

The next step is to generate a second generation population of solutions from those selected through genetic operators: crossover (also called recombination), and/or mutation.

For each new solution to be produced, a pair of "parent" solutions is selected for breeding from the pool selected previously. By producing a "child" solution using the above methods of crossover and mutation, a new solution is created which typically shares many of the characteristics of its "parents". New parents are selected for each child, and the process continues until a new population of solutions of appropriate size is generated. Although reproduction methods that are based on the use of two parents are more "biology inspired", recent researches (Islam Abou El Ata 2006) suggested more than two "parents" are better to be used to reproduce a good quality chromosome.

These processes ultimately result in the next generation population of chromosomes that is different from the initial generation. Generally the average fitness will have increased by this procedure for the population, since only the best organisms from the first generation are selected for breeding, along with a small proportion of less fit solutions, for reasons already mentioned above.

### **3.4 Termination**

This generational process is repeated until a termination condition has been reached. Common terminating conditions are:

- A solution is found that satisfies minimum criteria
- Fixed number of generations reached
- Allocated budget (computation time/money) reached
- The highest ranking solution's fitness is reaching or has reached a plateau such that successive iterations no longer produce better results
- Manual inspection
- Combinations of the above

### **3.5 Simple generational genetic algorithm pseudocode**

1. Choose initial population
2. Evaluate the fitness of each individual in the population
3. Repeat until termination: (time limit or sufficient fitness achieved)
  1. Select best-ranking individuals to reproduce
  2. Breed new generation through crossover and/or mutation (genetic operations) and give birth to offspring
  3. Evaluate the individual fitness of the offspring
  4. Replace worst ranked part of population with offspring

### **3.6 Flowchart**

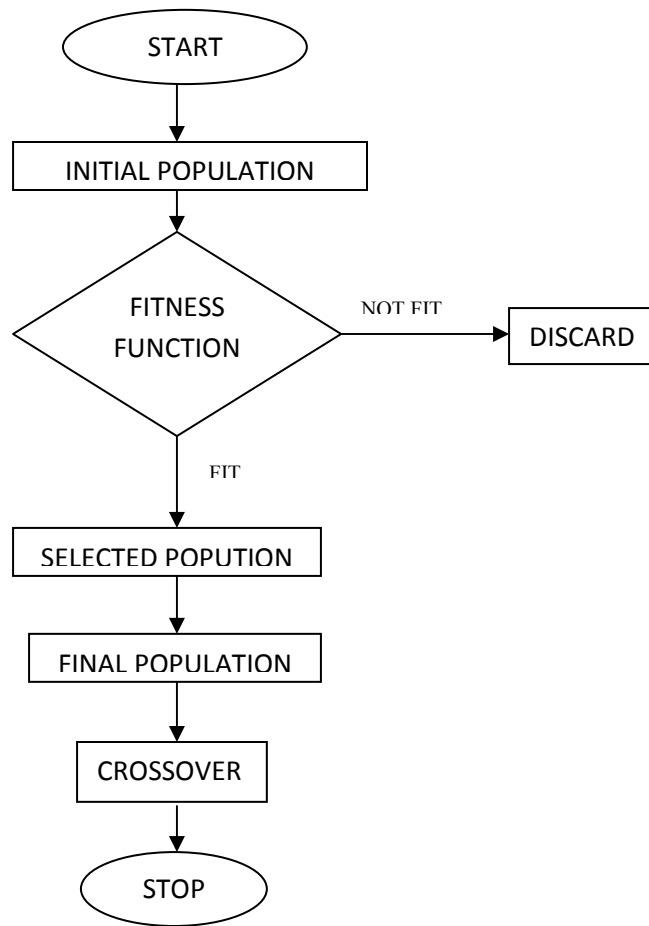


Fig 11

### **3.7 Observations**

There are several general observations about the generation of solutions via a genetic algorithm:

- Repeated fitness function evaluation for complex problems is often the most prohibitive and limiting segment of artificial evolutionary algorithms. Finding optimal solution to complex high dimensional, multimodal problems often requires very expensive fitness function evaluations. In real world problems such as structural optimization problems, one single function evaluation may require several hours to several days of complete simulation. Typical optimization method can not deal with such a type of problem. In this case, it may be necessary to forgo an exact evaluation and use an approximated fitness that is computationally efficient. It is apparent that amalgamation of approximate models

may be one of the most promising approaches to convincingly use EA to solve complex real life problems.

- The "better" is only in comparison to other solution. As a result, the stop criterion is not clear.
- In many problems, GAs may have a tendency to converge towards local optima or even arbitrary points rather than the global optimum of the problem. This means that it does not "know how" to sacrifice short-term fitness to gain longer-term fitness. The likelihood of this occurring depends on the shape of the fitness landscape: certain problems may provide an easy ascent towards a global optimum, others may make it easier for the function to find the local optima. This problem may be alleviated by using a different fitness function, increasing the rate of mutation, or by using selection techniques that maintain a diverse population of solutions, although the No Free Lunch theorem proves that there is no general solution to this problem. A common technique to maintain diversity is to impose a "niche penalty", wherein, any group of individuals of sufficient similarity (niche radius) have a penalty added, which will reduce the representation of that group in subsequent generations, permitting other (less similar) individuals to be maintained in the population. This trick, however, may not be effective, depending on the landscape of the problem. Diversity is important in genetic algorithms (and genetic programming) because crossing over a homogeneous population does not yield new solutions. In evolution strategies and evolutionary programming, diversity is not essential because of a greater reliance on mutation.
- Operating on dynamic data sets is difficult, as genomes begin to converge early on towards solutions which may no longer be valid for later data. Several methods have been proposed to remedy this by increasing genetic diversity somehow and preventing early convergence, either by increasing the probability of mutation when the solution quality drops (called *triggered hypermutation*), or by occasionally introducing entirely new, randomly generated elements into the gene pool (called *random immigrants*). Again, evolution strategies and evolutionary programming can be implemented with a so-called "comma strategy" in which parents are not maintained and new parents are selected only from offspring. This can be more effective on dynamic problems.



- GAs cannot effectively solve problems in which the only fitness measure is a single right/wrong measure, as there is no way to converge on the solution (no hill to climb). In these cases, a random search may find a solution as quickly as a GA. *However*, if the situation allows the success/failure trial to be repeated giving (possibly) different results, then the ratio of successes to failures provides a suitable fitness measure.
- Selection is clearly an important genetic operator, but opinion is divided over the importance of crossover versus mutation. Some argue that crossover is the most important, while mutation is only necessary to ensure that potential solutions are not lost. Others argue that crossover in a largely uniform population only serves to propagate innovations originally found by mutation, and in a non-uniform population crossover is nearly always equivalent to a very large mutation (which is likely to be catastrophic). There are many references in Fogel (2006) that support the importance of mutation-based search, but across all problems the No Free Lunch theorem holds, so these opinions are without merit unless the discussion is restricted to a particular problem.
- Often, GAs can rapidly locate *good* solutions, even for difficult search spaces. The same is of course also true for evolution strategies and evolutionary programming.
- For specific optimization problems and problem instances, other optimization algorithms may find better solutions than genetic algorithms (given the same amount of computation time). Alternative and complementary algorithms include evolution strategies, evolutionary programming, simulated annealing, Gaussian adaptation, hill climbing, and swarm intelligence (e.g.: ant colony optimization, particle swarm optimization) and methods based on integer linear programming. The question of which, if any, problems are suited to genetic algorithms (in the sense that such algorithms are better than others) is open and controversial.
- As with all current machine learning problems it is worth tuning the parameters such as mutation probability, recombination probability and population size to find reasonable settings for the problem class being worked on. A very small mutation rate may lead to genetic drift (which is non-ergodic in nature). A recombination rate that is too high may lead to premature convergence of the genetic algorithm. A mutation rate that is too high may lead to loss of good solutions unless there is elitist selection. There are theoretical

but not yet practical upper and lower bounds for these parameters that can help guide selection.

- The implementation and evaluation of the fitness function is an important factor in the speed and efficiency of the algorithm.

### **3.8 Variants**

The simplest algorithm represents each chromosome as a bit string. Typically, numeric parameters can be represented by integers, though it is possible to use floating point representations. The floating point representation is natural to evolution strategies and evolutionary programming. The notion of real-valued genetic algorithms has been offered but is really a misnomer because it does not really represent the building block theory that was proposed by Holland in the 1970s. This theory is not without support though, based on theoretical and experimental results (see below). The basic algorithm performs crossover and mutation at the bit level. Other variants treat the chromosome as a list of numbers which are indexes into an instruction table, nodes in a linked list, hashes, objects, or any other imaginable data structure. Crossover and mutation are performed so as to respect data element boundaries. For most data types, specific variation operators can be designed. Different chromosomal data types seem to work better or worse for different specific problem domains.

When bit strings representations of integers are used, Gray coding is often employed. In this way, small changes in the integer can be readily effected through mutations or crossovers. This has been found to help prevent premature convergence at so called *Hamming walls*, in which too many simultaneous mutations (or crossover events) must occur in order to change the chromosome to a better solution.

Other approaches involve using arrays of real-valued numbers instead of bit strings to represent chromosomes. Theoretically, the smaller the alphabet, the better the performance, but paradoxically, good results have been obtained from using real-valued chromosomes.

A very successful (slight) variant of the general process of constructing a new population is to allow some of the better organisms from the current generation to carry over to the next, unaltered. This strategy is known as *elitist selection*.

Parallel implementations of genetic algorithms come in two flavours. Coarse grained parallel genetic algorithms assume a population on each of the computer nodes and migration of individuals among the nodes. Fine grained parallel genetic algorithms assume an individual on each processor node which acts with neighboring individuals for selection and reproduction. Other variants, like genetic algorithms for online optimization problems, introduce time-dependence or noise in the fitness function.

It can be quite effective to combine GA with other optimization methods. GA tends to be quite good at finding generally good global solutions, but quite inefficient at finding the last few mutations to find the absolute optimum. Other techniques (such as simple hill climbing) are quite efficient at finding absolute optimum in a limited region. Alternating GA and hill climbing can improve the efficiency of GA while overcoming the lack of robustness of hill climbing.

This means that the rules of genetic variation may have a different meaning in the natural case. For instance - provided that steps are stored in consecutive order - crossing over may sum a number of steps from maternal DNA adding a number of steps from paternal DNA and so on. This is like adding vectors that more probably may follow a ridge in the phenotypic landscape. Thus, the efficiency of the process may be increased by many orders of magnitude. Moreover, the inversion operator has the opportunity to place steps in consecutive order or any other suitable order in favour of survival or efficiency. (See for instance or example in travelling salesman problem.)

Population-based incremental learning is a variation where the population as a whole is evolved rather than its individual members.

## CHAPTER – 4

### GENETIC K-MEANS ALGORITHM

#### **4.1 Why Genetic K-Means Algorithm?**

Genetic algorithm's (GA) used earlier in clustering employ either an expensive crossover operator to generate valid child chromosomes from parent chromosomes or a costly fitness function or both. To circumvent these expensive operations, we hybridize GA with a classical gradient descent algorithm used in clustering viz., K-means algorithm. Hence, the name Genetic K-means algorithm (GKA). We define K-means operator, one-step of K-means algorithm, and use it in GKA as a search operator instead of mutation. We prove that the GKA converges to the global optimum. It is observed in the simulations that GKA converges to the best known optimum corresponding to the given data in concurrence with the convergence result. It is also observed that GKA searches faster than some of the other algorithms used for clustering. Since, K-means algorithm uses the entire dataset for clustering. GKA improved predictability of the system since the final population considered for clustering has the maximum probability.

#### **4.2 Introduction**

GA's have been applied to many function optimization problems and are shown to be good in finding optimal and near optimal solutions. Their robustness of search in large search spaces and their domain independent nature motivated their applications in various fields like pattern recognition, machine learning, VLSI design, etc.

The simplest and most popular among iterative clustering algorithms is the K-means algorithm (KMA). As mentioned above, this algorithm may converge to a suboptimal partition. Since stochastic optimization approaches are good at avoiding convergence to a locally optimal solution, these approaches could be used to find a globally optimal solution. One of the important problems in partitional clustering is to find a partition of the given data, with a specified number of clusters, that minimizes the total within cluster variation. To prove that it converges to the global optimum and compare its performance with original K-Means algorithm.

Genetic algorithms (GA's) work on a coding of the parameter set over which the search has to be performed, rather than the parameters themselves. These encoded parameters are called *solutions* or *chromosomes* and the objective function value at a solution is the objective function value at the corresponding parameters. GA's solve optimization problems using a population of a fixed number, called the *population size*, of solutions. A solution consists of a string of symbols, typically binary symbols. GA's evolve over *generations*. During each generation, they produce a new population from the current population by applying genetic operators viz., *natural selection*, *crossover*, and *mutation*. Each solution in the population is associated with a figure of merit (fitness value) depending on the value of the function to be optimized. The selection operator selects a solution from the current population for the next population with probability proportional to its fitness value. Crossover operates on two solution strings and results in another two strings. Typical crossover operator exchange the segments of selected strings across a crossover point with a probability. Recently, it has been shown that the GA's that maintain the best discovered solution either before or after the selection operator asymptotically converge to the global optimum. Thus, GKA combines the simplicity of the K-means algorithm and the robust nature of GA's.

#### **4.3 Methodology**

GKA maintains a population of coded solutions. The population is initialized randomly and is evolved over generations. The population in the next generation is obtained by applying genetic Operators on the current population. The evolution takes place until a terminating condition is reached. The genetic operators that are used in GKA are the selection, fitness function, crossover and the K-means operator.

- *Initialization:* since it is a forest fires prediction system. So, the database is forest fires from machine learning repository with 13 attributes. Initially the entire database is considered as an input to GKA and is connected to the algorithm through JDBC-ODBC.
- *Selection:* The selection process selects chromosome from the previous population. Solutions in the current population are evaluated based on their merit to survive in the next population.

This requires that each solution in a population be associated with a figure of merit or fitness value. The fitness value in this case is determined on the basis of fitness function which is given as follows:

fitness function= $(.732)*data[1]+(.044)*(data[2]+data[3])+(.18)*data[4]$

where data[1] =temperature

data[2]=RH

data [3]=wind

data [4]=Rain

The sensitivity analysis of forest fires show the given percentage of these four attributes which contribute to forest fires

**Temp**= 73.2%

**RH**= 4.1%

**Wind**=4.7%

**Rain**=18.0%

So, population is selected on basis of above fitness function. First, a mid value 15 is considered and all entries below 15 are discarded to get the selected population.

- *Crossover* : The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size, which facilitates simple crossover operation. In this case crossover is carried out on every two rows to reproduce new rows.
- *K-means* : After crossover, fitness function is applied again to get the final population on which K-means is applied to get the clusters of the location where the probability of forest fires is maximum.

#### 4.5 Flowchart

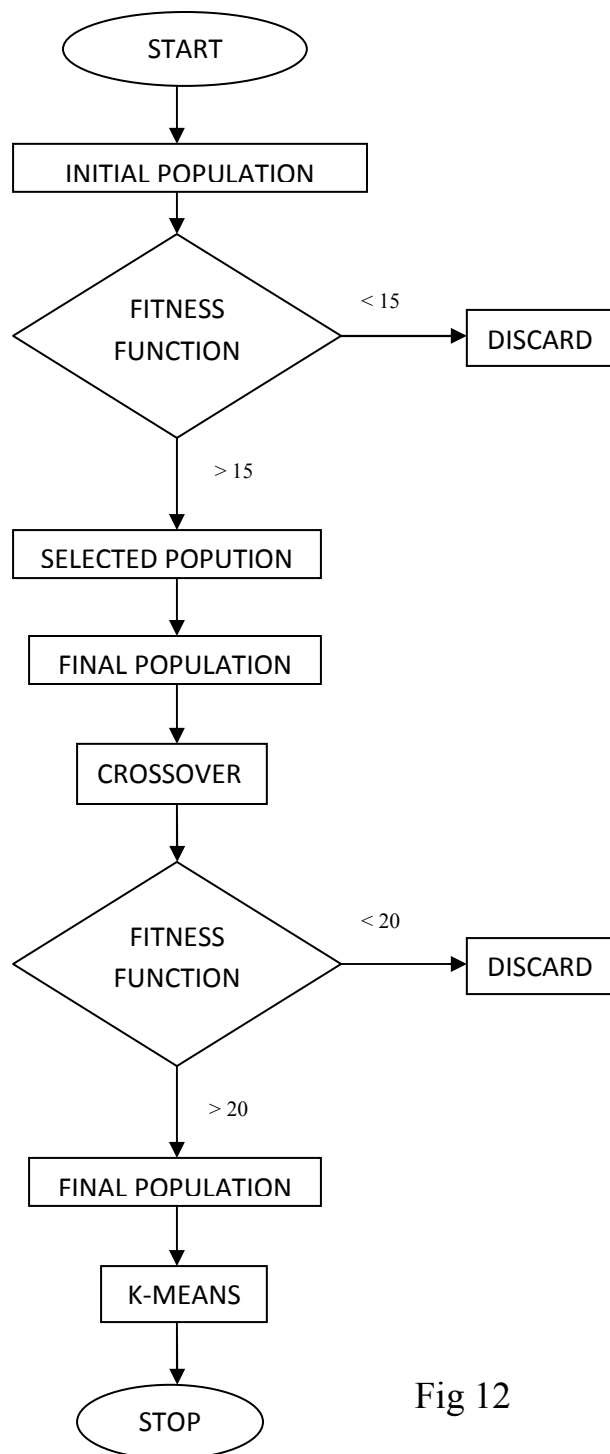


Fig 12

## CHAPTER – 5

### CODE IMPLEMENTATION

#### 5.1 Centroid

```
/**
```

```
* This class represents the Centroid for a Cluster. The initial centroid is calculated  
* using a equation which divides the sample space for each dimension into equal parts  
* depending upon the value of k.  
*/
```

```
class Centroid
```

```
{
```

```
    private double mCx2,mCx3,mCx4,mCx5,mCx6,mCx7,mCx8,mCx9,mCx10,mCx11,mCx12;
```

```
    private Cluster mCluster;
```

```
    public Centroid(double cx2, double cx3, double cx4, double cx5, double cx6, double cx7,  
        double cx8, double cx9, double cx10, double cx11, double cx12)
```

```
    {
```

```
        this.mCx2 = cx2;
```

```
        this.mCx3 = cx3;
```

```
        this.mCx4 = cx4;
```

```
        this.mCx5 = cx5;
```

```
        this.mCx6 = cx6;
```

```
        this.mCx7 = cx7;
```

```
        this.mCx8 = cx8;
```

```
        this.mCx9 = cx9;
```

```
        this.mCx10 = cx10;
```

```
        this.mCx11 = cx11;
```

```
        this.mCx12 = cx12;
```

```
    }
```



```

public void calcCentroid()
{
    //only called by CAInstance
    int numDP = mCluster.getNumDataPoints();
    double tempX2 = 0, tempX3 = 0, tempX4 = 0, tempX5 = 0, tempX6 = 0,
    tempX7 = 0, tempX8 = 0, tempX9 = 0, tempX10 = 0, tempX11 = 0,
    tempX12 = 0;
    int i;
    //caluclating the new Centroid
    for (i = 0; i < numDP; i++)
    {
        tempX2 = tempX2 + mCluster.getDataPoint(i).getX2();
        tempX3 = tempX3 + mCluster.getDataPoint(i).getX3();
        tempX4 = tempX4 + mCluster.getDataPoint(i).getX4();
        tempX5 = tempX5 + mCluster.getDataPoint(i).getX5();
        tempX6 = tempX6 + mCluster.getDataPoint(i).getX6();
        tempX7 = tempX7 + mCluster.getDataPoint(i).getX7();
        tempX8 = tempX8 + mCluster.getDataPoint(i).getX8();
        tempX9 = tempX9 + mCluster.getDataPoint(i).getX9();
        tempX10 = tempX10 + mCluster.getDataPoint(i).getX10();
        tempX11 = tempX11 + mCluster.getDataPoint(i).getX11();
        tempX12 = tempX12 + mCluster.getDataPoint(i).getX12();
    }

    this.mCx2 = tempX2 / numDP;
    this.mCx3 = tempX3 / numDP;
    this.mCx4 = tempX4 / numDP;
    this.mCx5 = tempX5 / numDP;
    this.mCx6 = tempX6 / numDP;
    this.mCx7 = tempX7 / numDP;
    this.mCx8 = tempX8 / numDP;

```

```

        this.mCx9 = tempX9 / numDP;
        this.mCx10 = tempX10 / numDP;
        this.mCx11 = tempX11 / numDP;
        this.mCx12 = tempX12 / numDP;

        //calculating the new Euclidean Distance for each Data Point
        tempX2 = 0;
        tempX3 = 0;
        tempX4 = 0;
        tempX5 = 0;
        tempX6 = 0;
        tempX7 = 0;
        tempX8 = 0;
        tempX9 = 0;
        tempX10 = 0;
        tempX11 = 0;
        tempX12 = 0;
        for (i = 0; i < numDP; i++)
        {
            mCluster.getDataPoint(i).calcEuclideanDistance();
        }

        //calculate the new Sum of Squares for the Cluster
        mCluster.calcSumOfSquares();
    }

    public void setCluster(Cluster c)
    {
        this.mCluster = c;
    }

    public double getCx2()

```

```
{  
    return mCx2;  
}  
public double getCx3()  
{  
    return mCx3;  
}  
public double getCx4()  
{  
    return mCx4;  
}  
public double getCx5()  
{  
    return mCx5;  
}  
public double getCx6()  
{  
    return mCx6;  
}  
public double getCx7()  
{  
    return mCx7;  
}  
public double getCx8()  
{  
    return mCx8;  
}  
public double getCx9()  
{  
    return mCx9;  
}
```

```

    public double getCx10()
    {
        return mCx10;
    }
    public double getCx11()
    {
        return mCx11;
    }
    public double getCx12()
    {
        return mCx12;
    }
    public Cluster getCluster()
    {
        return mCluster;
    }
}

```

## **5.2 Cluster**

```
import java.util.Vector;
```

```
/**
```

```

 * This class represents a Cluster in a Cluster Analysis Instance. A Cluster is associated
 * with one and only one JCA Instance. A Cluster is related to more than one DataPoints and
 * one centroid.

```

```
*/
```

```
class Cluster
{
    private String mName;
    private Centroid mCentroid;
    private double mSumSqr;
    private Vector mDataPoints;

    public Cluster(String name)
    {
        this.mName = name;
        this.mCentroid = null; //will be set by calling setCentroid()
        mDataPoints = new Vector();
    }
    public void setCentroid(Centroid c)
    {
        mCentroid = c;
    }
    public Centroid getCentroid()
    {
        return mCentroid;
    }
    public void addDataPoint(DataPoint dp)
    {
        //called from CAInstance
        dp.setCluster(this); //initiates a inner call to calcEuclideanDistance() in DP.
        this.mDataPoints.addElement(dp);
        calcSumOfSquares();
    }
    public void removeDataPoint(DataPoint dp)
    {
        this.mDataPoints.removeElement(dp);
    }
}
```

```

        calcSumOfSquares();
    }
    public int getNumDataPoints()
    {
        return this.mDataPoints.size();
    }
    public DataPoint getDataPoint(int pos)
    {
        return (DataPoint) this.mDataPoints.elementAt(pos);
    }
    public void calcSumOfSquares()
    {
        //called from Centroid
        int size = this.mDataPoints.size();
        double temp = 0;
        for (int i = 0; i < size; i++)
        {
            temp = temp + ((DataPoint)this.mDataPoints.elementAt(i)).getCurrentEuDt();
        }
        this.mSumSqr = temp;
    }
    public double getSumSqr()
    {
        return this.mSumSqr;
    }
    public String getName()
    {
        return this.mName;
    }
    public Vector getDataPoints()
    {

```

```

        return this.mDataPoints;
    }

}

```

### **5.3 Data Points**

```
/**
```

This class represents a candidate for Cluster analysis. A candidate must have a name and two independent variables on the basis of which it is to be clustered. A Data Point must have two variables and a name. A Vector of Data Point object is fed into the constructor of the JCA class. JCA and DataPoint are the only classes which may be available from other packages.

```
*/
```

```

public class DataPoint
{
    private double mX2,mX3,mX4,mX5,mX6,mX7,mX8,mX9,mX10,mX11,mX12;
    private String mObjName;
    private Cluster mCluster;
    private double mEuDt;

    public DataPoint(double x2, double x3, double x4,double x5, double x6,
double x7,double x8, double x9, double x10,double x11, double x12, String name)
    {
        this.mX2 = x2;
        this.mX3 = x3;
        this.mX4 = x4;
        this.mX5 = x5;
        this.mX6 = x6;
        this.mX7 = x7;
    }
}

```

```

        this.mX8 = x8;
        this.mX9 = x9;
        this.mX10 = x10;
        this.mX11 = x11;
        this.mX12 = x12;
        this.mObjName = name;
        this.mCluster = null;
    }
    public void setCluster(Cluster cluster)
    {
        this.mCluster = cluster;
        calcEuclideanDistance();
    }
    public void calcEuclideanDistance()
    {
        //called when DP is added to a cluster or when a Centroid is recalculated.
        mEuDt = Math.sqrt(Math.pow((mX2 - mCluster.getCentroid().getCx2()),2) +
        Math.pow((mX3 - mCluster.getCentroid().getCx3()), 2) + Math.pow((mX4 -
        mCluster.getCentroid().getCx4()),2)+Math.pow((mX5 -
        mCluster.getCentroid().getCx5()),2) + Math.pow((mX6 -
        mCluster.getCentroid().getCx6()), 2) + Math.pow((mX7 -
        mCluster.getCentroid().getCx7()),2)+Math.pow((mX8 -
        mCluster.getCentroid().getCx8()),2) + Math.pow((mX9 -
        mCluster.getCentroid().getCx9()), 2) + Math.pow((mX10 -
        mCluster.getCentroid().getCx10()),2)+Math.pow((mX11 -
        mCluster.getCentroid().getCx11()),2) + Math.pow((mX12 -
        mCluster.getCentroid().getCx12()), 2));
    }

```



```

public double testEuclideanDistance(Centroid c)
{
    return Math.sqrt(Math.pow((mX2 - mCluster.getCentroid().getCx2()),2) +
        Math.pow((mX3 - mCluster.getCentroid().getCx3()), 2) + Math.pow((mX4 -
        mCluster.getCentroid().getCx4()),2)+Math.pow((mX5 -
        mCluster.getCentroid().getCx5()),2) + Math.pow((mX6 -
        mCluster.getCentroid().getCx6()), 2) + Math.pow((mX7 -
        mCluster.getCentroid().getCx7()),2)+Math.pow((mX8 -
        mCluster.getCentroid().getCx8()),2) + Math.pow((mX9 -
        mCluster.getCentroid().getCx9()), 2) + Math.pow((mX10 -
        mCluster.getCentroid().getCx10()),2)+Math.pow((mX11 -
        mCluster.getCentroid().getCx11()),2) + Math.pow((mX12 -
        mCluster.getCentroid().getCx12()), 2));
}

public double getX2()
{
    return mX2;
}

public double getX3()
{
    return mX3;
}

public double getX4()
{
    return mX4;
}

public double getX5()
{
    return mX5;
}

```

```
public double getX6()
{
    return mX6;
}
public double getX7()
{
    return mX7;
}
public double getX8()
{
    return mX8;
}
public double getX9()
{
    return mX9;
}
public double getX10()
{
    return mX10;
}
public double getX11()
{
    return mX11;
}
public double getX12()
{
    return mX12;
}
public Cluster getCluster()
{
    return mCluster;
}
```

```

    }
    public double getCurrentEuDt()
    {
        return mEuDt;
    }
    public String getObjName()
    {
        return mObjName;
    }
}

```

#### **5.4 GA Implementation**

```

import java.io.*;
import java.sql.*;

class Ga
{
    public int fitness(String db,String tb,int value)
    {
        double temp=0;
        int count=0,t=0;
        double data[]=new double[5];
        try
        {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection A;
            A=DriverManager.getConnection("jdbc:odbc:"+db);
            Statement stmt=A.createStatement();
            PreparedStatement sStmt=A.prepareStatement("Delete from f where id=?");
            ResultSet rs=stmt.executeQuery("Select * from "+tb);

```

```

while(rs.next())
{
    data[0]=Double.parseDouble(rs.getString(1));
    data[1]=Double.parseDouble(rs.getString(8));
    data[2]=Double.parseDouble(rs.getString(9));
    data[3]=Double.parseDouble(rs.getString(10));
    data[4]=Double.parseDouble(rs.getString(11));
    temp=(.732)*data[1]+(.044)*(data[2]+data[3])+(.18)*data[4];
    if(temp>value)
    {
        count++;
        //System.out.println(data[0]+"--->"+"1");
        /*sStmt=A.prepareStatement("Insert into "+tb+"(id) values(?)
        where id=?");
        sStmt.setInt(1,count);
        sStmt.setDouble(2,data[0]);
        sStmt.executeUpdate();
    }
    Else
    {
        //System.out.println(data[0]+"--->"+"0");
        sStmt=A.prepareStatement("Delete from "+tb+" where id=?");
        sStmt.setDouble(1,data[0]);
        sStmt.executeUpdate();
    }
}
stmt.close();
sStmt.close();
A.close();
}

```

```

        catch(Exception e)
        {
            System.out.println(e);
        }
        //System.out.println(count);
        return count;
    }
    public int crossover(String db,String tb)
    {
        Try
        {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            int co=518;
            Connection A;
            A=DriverManager.getConnection("jdbc:odbc:"+db);
            Statement stmt=A.createStatement();
            Statement stmt1=A.createStatement();
            PreparedStatement sStmt=A.prepareStatement("Delete from "+tb+" where id=?");
            ResultSet rs=stmt.executeQuery("Select * from "+tb+" where id<517");
            ResultSet rs1=stmt1.executeQuery("Select * from "+tb+" where id<517");
            rs1.next();
            while(rs.next())
            {
                if(rs1.next())
                {
                    sStmt=A.prepareStatement("Insert into "+tb+" values(?,?,?,?,?,?,?,?,?,?,?)");
                    sStmt.setInt(1,co++);
                    sStmt.setString(2,rs.getString(2));
                    sStmt.setString(3,rs.getString(3));
                    sStmt.setString(4,rs.getString(4));
                    sStmt.setString(5,rs.getString(5));

```

```

        sStmt.setString(6,rs.getString(6));
        sStmt.setString(7,rs.getString(7));
sStmt.setDouble(8,(Double.parseDouble(rs.getString(8))+Double.parseDouble(rs1.getString(8)))
/2);
sStmt.setDouble(9,(Double.parseDouble(rs.getString(9))+Double.parseDouble(rs1.getString(9)))
/2);
sStmt.setDouble(10,(Double.parseDouble(rs.getString(10))+Double.parseDouble(rs1.getString(1
0))))/2);
sStmt.setDouble(11,(Double.parseDouble(rs.getString(11))+Double.parseDouble(rs1.getString(1
1))))/2);

        sStmt.setString(12,rs.getString(12));
        sStmt.setString(13,rs.getString(13));
        sStmt.setString(14,rs.getString(14));
        sStmt.executeUpdate();
    }
    Else
    {
sStmt=A.prepareStatement("Insert into "+tb+" values(?,?,?,?,?,?,?,?,?,?,?,?,?)");
        sStmt.setInt(1,co++);
        sStmt.setString(2,rs.getString(2));
        sStmt.setString(3,rs.getString(3));
        sStmt.setString(4,rs.getString(4));
        sStmt.setString(5,rs.getString(5));
        sStmt.setString(6,rs.getString(6));
        sStmt.setString(7,rs.getString(7));
sStmt.setDouble(8,(Double.parseDouble(rs.getString(8)))/2);
sStmt.setDouble(9,(Double.parseDouble(rs.getString(9)))/2);
sStmt.setDouble(10,(Double.parseDouble(rs.getString(10)))/2);
sStmt.setDouble(11,(Double.parseDouble(rs.getString(11)))/2);
        sStmt.setString(12,rs.getString(12));
        sStmt.setString(13,rs.getString(13));

```

```

        sStmt.setString(14,rs.getString(14));
        sStmt.executeUpdate();

    }

}

stmt.close();
sStmt.close();
A.close();

}
catch(Exception e)
{
    System.out.println(e);
}
return 0;
}

}

```

### **5.5 GA Main**

```

class GaMain
{
    public static void main(String args[])
    {
        Ga test=new Ga();
        test.fitness(15);
        test.crossover();
        test.fitness(20);
        System.out.println("-----End-----");
    }
}

```

```
}
```

## **5.6 Interface**

```
/*
 * InterFace.java
 *
 */

import java.sql.*;

/**
 *
 */

public class InterFace extends javax.swing.JFrame
{
    String db,tb;
    /** Creates new form InterFace */
    public InterFace()
    {
        initComponents();
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
    /** This method is called from within the constructor to
     * initialize the form.
     *
     * @param args the arguments to the constructor
     */
    // <editor-fold defaultstate="collapsed" desc=" Generated Code "> //GEN-BEGIN:
    initComponents

    private void initComponents()
    {
        jLabel1 = new javax.swing.JLabel();
    }
}
```



```

jLabel2 = new javax.swing.JLabel();
jTextField1 = new javax.swing.JTextField();
jLabel3 = new javax.swing.JLabel();
jTextField2 = new javax.swing.JTextField();
jLabel4 = new javax.swing.JLabel();
jScrollPane1 = new javax.swing.JScrollPane();
jList1 = new javax.swing.JList();
jButton1 = new javax.swing.JButton();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
jLabel1.setBackground(new java.awt.Color(204, 255, 255));
jLabel1.setFont(new java.awt.Font("Arial", 1, 24));
jLabel1.setForeground(new java.awt.Color(0, 204, 204));
jLabel1.setText("Forest Fire Prediction System");

jLabel2.setText("DataBase Name :");

jTextField1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jTextField1ActionPerformed(evt);
    }
});

jLabel3.setText("Table Name :");

jTextField2.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jTextField2ActionPerformed(evt);
    }
});

```

```

jLabel4.setText("Algorithm :");

jList1.setModel(new javax.swing.AbstractListModel() {
    String[] strings = { "K-Means", "Genetic Algorithm", "Hybrid" };
    public int getSize() { return strings.length; }
    public Object getElementAt(int i) { return strings[i]; }
});
jScrollPane1.setViewportViewView(jList1);

jButton1.setText("EXECUTE");
jButton1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton1ActionPerformed(evt);
    }
});

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
getContentPane().setLayout(layout);
layout.setHorizontalGroup(
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addGap(136, 136, 136)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(jLabel1)
                .addGroup(layout.createSequentialGroup()
                    .addGap(136, 136, 136)
                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                        .addComponent(jLabel2)
                        .addComponent(jLabel3)
                        .addComponent(jLabel4))
                )
            )
        )
);

```

```

        .addGap(42, 42, 42)

        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
            .addComponent(jScrollPane1)
            .addComponent(jTextField2)
            .addComponent(jTextField1, javax.swing.GroupLayout.DEFAULT_SIZE, 197,
Short.MAX_VALUE)
            .addComponent(jButton1))))
        .addContainerGap(140, Short.MAX_VALUE))
    );
    layout.setVerticalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addContainerGap()
            .addComponent(jLabel1)
            .addGap(46, 46, 46)

        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel2)
            .addComponent(jTextField1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(44, 44, 44)

        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel3)
            .addComponent(jTextField2, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(47, 47, 47)

        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jLabel4)

```

```

        .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 22,
javax.swing.GroupLayout.PREFERRED_SIZE))
        .addGap(65, 65, 65)
        .addComponent(jButton1)
        .addGap(68, 68, 68))
    );
    pack();
} // </editor-fold> // GEN-END: initComponents

```

```

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) { // GEN-
FIRST:event_jButton1ActionPerformed
    //jTextField1.getText()+jTextField2.getText()+jList1.getSelectedIndex();
    new PrgMain().Hybrid(jTextField1.getText(),jTextField2.getText());
} // GEN-LAST:event_jButton1ActionPerformed

```

```

private void jTextField2ActionPerformed(java.awt.event.ActionEvent evt) { // GEN-
FIRST:event_jTextField2ActionPerformed
    tb=evt.getActionCommand();
} // GEN-LAST:event_jTextField2ActionPerformed

```

```

private void jTextField1ActionPerformed(java.awt.event.ActionEvent evt) { // GEN-
FIRST:event_jTextField1ActionPerformed
    db=evt.getActionCommand();
} // GEN-LAST:event_jTextField1ActionPerformed

```

```

// Variables declaration - do not modify // GEN-BEGIN:variables

```

```

private javax.swing.JButton jButton1;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;

```

```

private javax.swing.JList jList1;
private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JTextField jTextField1;
private javax.swing.JTextField jTextField2;
// End of variables declaration//GEN-END:variables

```

```

}

```

## **5.7 JCA**

```

import java.util.Vector;

```

```

/**

```

This class is the entry point for constructing Cluster Analysis objects. Each instance of JCA object is associated with one or more clusters, and a Vector of DataPoint objects. The JCA and DataPoint classes are the only classes available from other packages.

```

**/

```

```

public class JCA {
    private Cluster[] clusters;
    private int miter;
    private Vector mDataPoints = new Vector();
    private double mSWCSS;

    public JCA(int k, int iter, Vector dataPoints) {
        clusters = new Cluster[k];
        for (int i = 0; i < k; i++) {
            clusters[i] = new Cluster("Cluster" + i);
        }
    }

```

```

    this.miter = iter;
    this.mDataPoints = dataPoints;
}

private void calcSWCSS() {
    double temp = 0;
    for (int i = 0; i < clusters.length; i++) {
        temp = temp + clusters[i].getSumSqr();
    }
    mSWCSS = temp;
}

public void startAnalysis() {
    //set Starting centroid positions - Start of Step 1
    setInitialCentroids();
    int n = 0;
    //assign DataPoint to clusters
    loop1: while (true) {
        for (int l = 0; l < clusters.length; l++)
        {
            clusters[l].addDataPoint((DataPoint)mDataPoints.elementAt(n));
            n++;
            if (n >= mDataPoints.size())
                break loop1;
        }
    }

    //calculate E for all the clusters
    calcSWCSS();

    //recalculate Cluster centroids - Start of Step 2

```

```

for (int i = 0; i < clusters.length; i++) {
    clusters[i].getCentroid().calcCentroid();
}

//recalculate E for all the clusters
calcSWCSS();

for (int i = 0; i < miter; i++) {
    //enter the loop for cluster 1
    for (int j = 0; j < clusters.length; j++) {
        for (int k = 0; k < clusters[j].getNumDataPoints(); k++) {

            //pick the first element of the first cluster
            //get the current Euclidean distance
            double tempEuDt = clusters[j].getDataPoint(k).getCurrentEuDt();
            Cluster tempCluster = null;
            boolean matchFoundFlag = false;

            //call testEuclidean distance for all clusters
            for (int l = 0; l < clusters.length; l++) {

                //if testEuclidean < currentEuclidean then
                if (tempEuDt >
clusters[j].getDataPoint(k).testEuclideanDistance(clusters[l].getCentroid())) {
                    tempEuDt =
clusters[j].getDataPoint(k).testEuclideanDistance(clusters[l].getCentroid());
                    tempCluster = clusters[l];
                    matchFoundFlag = true;
                }
                //if statement - Check whether the Last EuDt is > Present EuDt

```

```

    }
    //for variable 'l' - Looping between different Clusters for matching a Data Point.
    //add DataPoint to the cluster and calcSWCSS

    if (matchFoundFlag) {
        tempCluster.addDataPoint(clusters[j].getDataPoint(k));
        clusters[j].removeDataPoint(clusters[j].getDataPoint(k));
        for (int m = 0; m < clusters.length; m++) {
            clusters[m].getCentroid().calcCentroid();
        }

        //for variable 'm' - Recalculating centroids for all Clusters

        calcSWCSS();
    }

    //if statement - A Data Point is eligible for transfer between Clusters.
    }
    //for variable 'k' - Looping through all Data Points of the current Cluster.
    } //for variable 'j' - Looping through all the Clusters.
    } //for variable 'i' - Number of iterations.
}

public Vector[] getClusterOutput() {
    Vector v[] = new Vector[clusters.length];
    for (int i = 0; i < clusters.length; i++) {
        v[i] = clusters[i].getDataPoints();
    }
    return v;
}

```



```

private void setInitialCentroids() {
    //kn = (round((max-min)/k)*n)+min where n is from 0 to (k-1).
    double cx2 = 0, cx3 = 0, cx4 = 0, cx5 = 0, cx6 = 0, cx7 = 0, cx8 = 0, cx9 = 0, cx10 = 0, cx11
= 0, cx12 = 0;
    for (int n = 1; n <= clusters.length; n++) {
        cx2 = (((getMaxX2Value() - getMinX2Value()) / (clusters.length + 1)) * n) +
getMinX2Value();
        cx3 = (((getMaxX3Value() - getMinX3Value()) / (clusters.length + 1)) * n) +
getMinX3Value();
        cx4 = (((getMaxX4Value() - getMinX4Value()) / (clusters.length + 1)) * n) +
getMinX4Value();
        cx5 = (((getMaxX5Value() - getMinX5Value()) / (clusters.length + 1)) * n) +
getMinX5Value();
        cx6 = (((getMaxX6Value() - getMinX6Value()) / (clusters.length + 1)) * n) +
getMinX6Value();
        cx7 = (((getMaxX7Value() - getMinX7Value()) / (clusters.length + 1)) * n) +
getMinX7Value();
        cx8 = (((getMaxX8Value() - getMinX8Value()) / (clusters.length + 1)) * n) +
getMinX8Value();
        cx9 = (((getMaxX9Value() - getMinX9Value()) / (clusters.length + 1)) * n) +
getMinX9Value();
        cx10 = (((getMaxX10Value() - getMinX10Value()) / (clusters.length + 1)) * n) +
getMinX10Value();
        cx11 = (((getMaxX11Value() - getMinX11Value()) / (clusters.length + 1)) * n) +
getMinX11Value();
        cx12 = (((getMaxX12Value() - getMinX12Value()) / (clusters.length + 1)) * n) +
getMinX12Value();
        Centroid c1 = new Centroid(cx2, cx3, cx4, cx5, cx6, cx7, cx8, cx9, cx10, cx11, cx12);
        clusters[n - 1].setCentroid(c1);
        c1.setCluster(clusters[n - 1]);
    }
}

```

```

    }
}

```

```

private double getMaxX2Value() {
    double temp;
    temp = ((DataPoint) mDataPoints.elementAt(0)).getX2();
    for (int i = 0; i < mDataPoints.size(); i++) {
        DataPoint dp = (DataPoint) mDataPoints.elementAt(i);
        temp = (dp.getX2() > temp) ? dp.getX2() : temp;
    }
    return temp;
}

```

```

private double getMinX2Value() {
    double temp = 0;
    temp = ((DataPoint) mDataPoints.elementAt(0)).getX2();
    for (int i = 0; i < mDataPoints.size(); i++) {
        DataPoint dp = (DataPoint) mDataPoints.elementAt(i);
        temp = (dp.getX2() < temp) ? dp.getX2() : temp;
    }
    return temp;
}

```

```

private double getMaxX3Value() {
    double temp;
    temp = ((DataPoint) mDataPoints.elementAt(0)).getX3();
    for (int i = 0; i < mDataPoints.size(); i++) {
        DataPoint dp = (DataPoint) mDataPoints.elementAt(i);
        temp = (dp.getX3() > temp) ? dp.getX3() : temp;
    }
    return temp;
}

```

```
}
```

```
private double getMinX3Value() {
    double temp = 0;
    temp = ((DataPoint) mDataPoints.elementAt(0)).getX3();
    for (int i = 0; i < mDataPoints.size(); i++) {
        DataPoint dp = (DataPoint) mDataPoints.elementAt(i);
        temp = (dp.getX3() < temp) ? dp.getX3() : temp;
    }
    return temp;
}
```

```
private double getMaxX4Value() {
    double temp;
    temp = ((DataPoint) mDataPoints.elementAt(0)).getX4();
    for (int i = 0; i < mDataPoints.size(); i++) {
        DataPoint dp = (DataPoint) mDataPoints.elementAt(i);
        temp = (dp.getX4() > temp) ? dp.getX4() : temp;
    }
    return temp;
}
```

```
private double getMinX4Value() {
    double temp = 0;
    temp = ((DataPoint) mDataPoints.elementAt(0)).getX4();
    for (int i = 0; i < mDataPoints.size(); i++) {
        DataPoint dp = (DataPoint) mDataPoints.elementAt(i);
        temp = (dp.getX4() < temp) ? dp.getX4() : temp;
    }
    return temp;
}
```

```
private double getMaxX5Value() {
```

```

double temp;
temp = ((DataPoint) mDataPoints.elementAt(0)).getX5();
for (int i = 0; i < mDataPoints.size(); i++) {
    DataPoint dp = (DataPoint) mDataPoints.elementAt(i);
    temp = (dp.getX5() > temp) ? dp.getX5() : temp;
}
return temp;
}

```

```

private double getMinX5Value() {
    double temp = 0;
    temp = ((DataPoint) mDataPoints.elementAt(0)).getX5();
    for (int i = 0; i < mDataPoints.size(); i++) {
        DataPoint dp = (DataPoint) mDataPoints.elementAt(i);
        temp = (dp.getX5() < temp) ? dp.getX5() : temp;
    }
    return temp;
}

```

```

private double getMaxX6Value() {
    double temp;
    temp = ((DataPoint) mDataPoints.elementAt(0)).getX6();
    for (int i = 0; i < mDataPoints.size(); i++) {
        DataPoint dp = (DataPoint) mDataPoints.elementAt(i);
        temp = (dp.getX6() > temp) ? dp.getX6() : temp;
    }
    return temp;
}

```

```

private double getMinX6Value() {
    double temp = 0;
    temp = ((DataPoint) mDataPoints.elementAt(0)).getX6();

```

```

    for (int i = 0; i < mDataPoints.size(); i++) {
        DataPoint dp = (DataPoint) mDataPoints.elementAt(i);
        temp = (dp.getX6() < temp) ? dp.getX6() : temp;
    }
    return temp;
}

private double getMaxX7Value() {
    double temp;
    temp = ((DataPoint) mDataPoints.elementAt(0)).getX7();
    for (int i = 0; i < mDataPoints.size(); i++) {
        DataPoint dp = (DataPoint) mDataPoints.elementAt(i);
        temp = (dp.getX7() > temp) ? dp.getX7() : temp;
    }
    return temp;
}

private double getMinX7Value() {
    double temp = 0;
    temp = ((DataPoint) mDataPoints.elementAt(0)).getX7();
    for (int i = 0; i < mDataPoints.size(); i++) {
        DataPoint dp = (DataPoint) mDataPoints.elementAt(i);
        temp = (dp.getX7() < temp) ? dp.getX7() : temp;
    }
    return temp;
}

private double getMaxX8Value() {
    double temp;
    temp = ((DataPoint) mDataPoints.elementAt(0)).getX8();
    for (int i = 0; i < mDataPoints.size(); i++) {
        DataPoint dp = (DataPoint) mDataPoints.elementAt(i);
        temp = (dp.getX8() > temp) ? dp.getX8() : temp;
    }
}

```

```

    }
    return temp;
}

```

```

private double getMinX8Value() {
    double temp = 0;
    temp = ((DataPoint) mDataPoints.elementAt(0)).getX8();
    for (int i = 0; i < mDataPoints.size(); i++) {
        DataPoint dp = (DataPoint) mDataPoints.elementAt(i);
        temp = (dp.getX8() < temp) ? dp.getX8() : temp;
    }
    return temp;
}

```

```

private double getMaxX9Value() {
    double temp;
    temp = ((DataPoint) mDataPoints.elementAt(0)).getX9();
    for (int i = 0; i < mDataPoints.size(); i++) {
        DataPoint dp = (DataPoint) mDataPoints.elementAt(i);
        temp = (dp.getX9() > temp) ? dp.getX9() : temp;
    }
    return temp;
}

```

```

private double getMinX9Value() {
    double temp = 0;
    temp = ((DataPoint) mDataPoints.elementAt(0)).getX9();
    for (int i = 0; i < mDataPoints.size(); i++) {
        DataPoint dp = (DataPoint) mDataPoints.elementAt(i);
        temp = (dp.getX9() < temp) ? dp.getX9() : temp;
    }
}

```

```

        return temp;
    }

    private double getMaxX10Value() {
        double temp;
        temp = ((DataPoint) mDataPoints.elementAt(0)).getX10();
        for (int i = 0; i < mDataPoints.size(); i++) {
            DataPoint dp = (DataPoint) mDataPoints.elementAt(i);
            temp = (dp.getX10() > temp) ? dp.getX10() : temp;
        }
        return temp;
    }

    private double getMinX10Value() {
        double temp = 0;
        temp = ((DataPoint) mDataPoints.elementAt(0)).getX10();
        for (int i = 0; i < mDataPoints.size(); i++) {
            DataPoint dp = (DataPoint) mDataPoints.elementAt(i);
            temp = (dp.getX10() < temp) ? dp.getX10() : temp;
        }
        return temp;
    }

    private double getMaxX11Value() {
        double temp;
        temp = ((DataPoint) mDataPoints.elementAt(0)).getX11();
        for (int i = 0; i < mDataPoints.size(); i++) {
            DataPoint dp = (DataPoint) mDataPoints.elementAt(i);
            temp = (dp.getX11() > temp) ? dp.getX11() : temp;
        }
        return temp;
    }

```

```

private double getMinX11Value() {
    double temp = 0;
    temp = ((DataPoint) mDataPoints.elementAt(0)).getX11();
    for (int i = 0; i < mDataPoints.size(); i++) {
        DataPoint dp = (DataPoint) mDataPoints.elementAt(i);
        temp = (dp.getX11() < temp) ? dp.getX11() : temp;
    }
    return temp;
}

private double getMaxX12Value() {
    double temp;
    temp = ((DataPoint) mDataPoints.elementAt(0)).getX12();
    for (int i = 0; i < mDataPoints.size(); i++) {
        DataPoint dp = (DataPoint) mDataPoints.elementAt(i);
        temp = (dp.getX12() > temp) ? dp.getX12() : temp;
    }
    return temp;
}

private double getMinX12Value() {
    double temp = 0;
    temp = ((DataPoint) mDataPoints.elementAt(0)).getX12();
    for (int i = 0; i < mDataPoints.size(); i++) {
        DataPoint dp = (DataPoint) mDataPoints.elementAt(i);
        temp = (dp.getX12() < temp) ? dp.getX12() : temp;
    }
    return temp;
}

public int getKVvalue() {

```



```

        return clusters.length;
    }

    public int getIterations() {
        return miter;
    }

    public int getTotalDataPoints() {
        return mDataPoints.size();
    }

    public double getSWCSS() {
        return mSWCSS;
    }

    public Cluster getCluster(int pos) {
        return clusters[pos];
    }
}

```

### **5.8 Prg Main**

```

/*-----PrgMain.java-----*/
import java.io.*;
import java.sql.*;

import java.util.Vector;
import java.util.Iterator;

```

```

public class PrgMain {
    public void Hybrid(String db,String tb){
        Ga test=new Ga();
        test.fitness(db,tb,15);
        test.crossover(db,tb);
        test.fitness(db,tb,20);
        Vector dataPoints = new Vector();

        try{
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection A;
            A=DriverManager.getConnection("jdbc:odbc:"+db);
            Statement stmt=A.createStatement();
            ResultSet rs=stmt.executeQuery("Select * from "+tb);
            while(rs.next()){
                dataPoints.add(new
DataPoint(Double.parseDouble(rs.getString(2)),Double.parseDouble(rs.getString(3)),Double.par
seDouble(rs.getString(4)),Double.parseDouble(rs.getString(5)),Double.parseDouble(rs.getString
(6)),Double.parseDouble(rs.getString(7)),Double.parseDouble(rs.getString(8)),Double.parseDou
ble(rs.getString(9)),Double.parseDouble(rs.getString(10)),Double.parseDouble(rs.getString(11)),
Double.parseDouble(rs.getString(12)),rs.getString(1)));

            }
            stmt.close();
            A.close();
        }
        catch(Exception e){
            System.out.println(e);
        }

        JCA jca = new JCA(3,1000,dataPoints);
    }
}

```

```

jca.startAnalysis();

Vector[] v = jca.getClusterOutput();
for (int i=0; i<v.length; i++){
    Vector tempV = v[i];
    System.out.println("\n-----Cluster"+i+"-----\n");
    Iterator iter = tempV.iterator();
    while(iter.hasNext()){
        DataPoint dpTemp = (DataPoint)iter.next();
        try{
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection A;
            A=DriverManager.getConnection("jdbc:odbc:"+db);
            PreparedStatement sStmt=A.prepareStatement("Insert into Cluster
values(?,?,?)");

            sStmt.setInt(1,i);
            sStmt.setDouble(2,dpTemp.getX2());
            sStmt.setDouble(3,dpTemp.getX3());
            sStmt.executeUpdate();
        }
        catch(Exception e){
            System.out.println(e);
        }

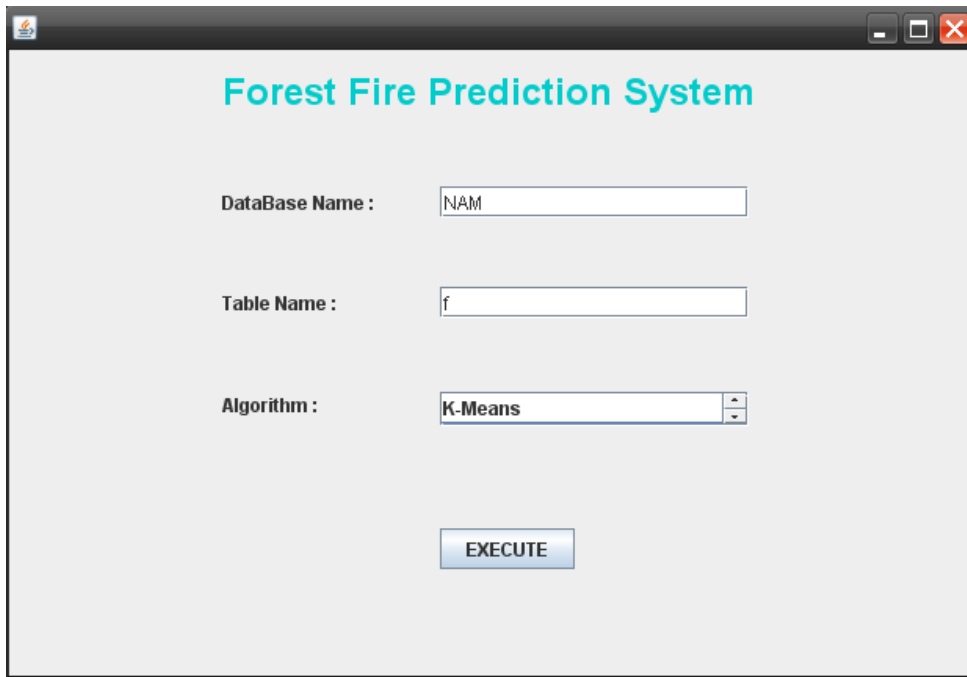
        System.out.println("[ "+dpTemp.getX4()+dpTemp.getX5()+", "+dpTemp.getX6()+", "+dp
Temp.getX7()+dpTemp.getX8()+", "+dpTemp.getX9()+", "+dpTemp.getX10()+dpTemp.getX11()
+", "+dpTemp.getX12()+"]"+" \tCordinates (x="+dpTemp.getX2()+", y="+dpTemp.getX3()+")");
    }
}

```

```
}  
}
```

## **5.9 Results**

### ***5.9.1 Snapshots With K-Means Selected***



The screenshot displays a software window titled "Forest Fire Prediction System". Inside the window, there are three input fields and one button. The first field is labeled "DataBase Name :" and contains the text "NAM". The second field is labeled "Table Name :" and contains the text "f". The third field is labeled "Algorithm :" and is a dropdown menu with "K-Means" selected. Below these fields is a blue button with the text "EXECUTE".

Fig 13

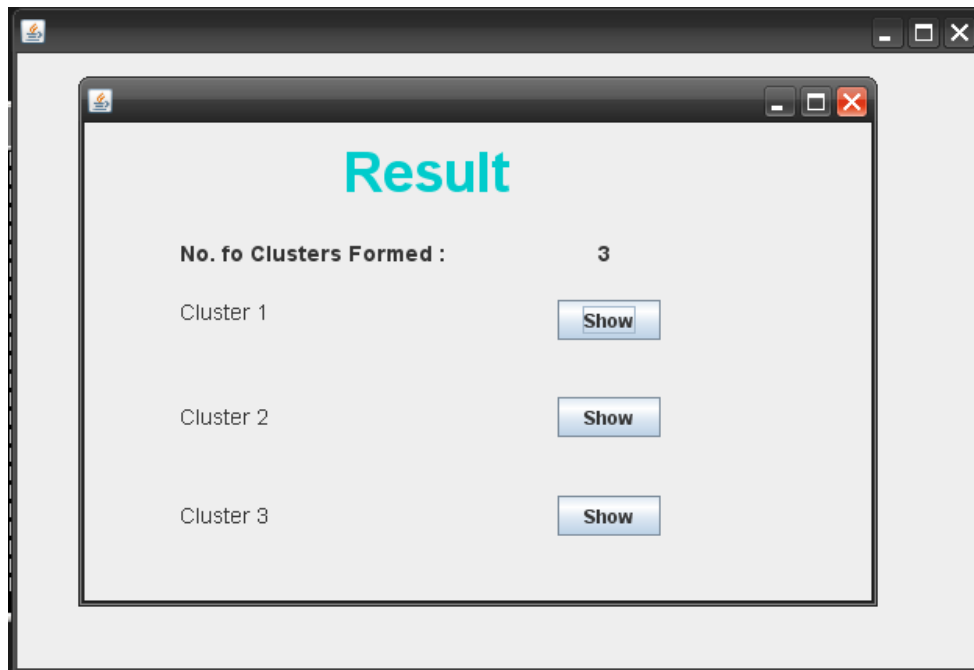
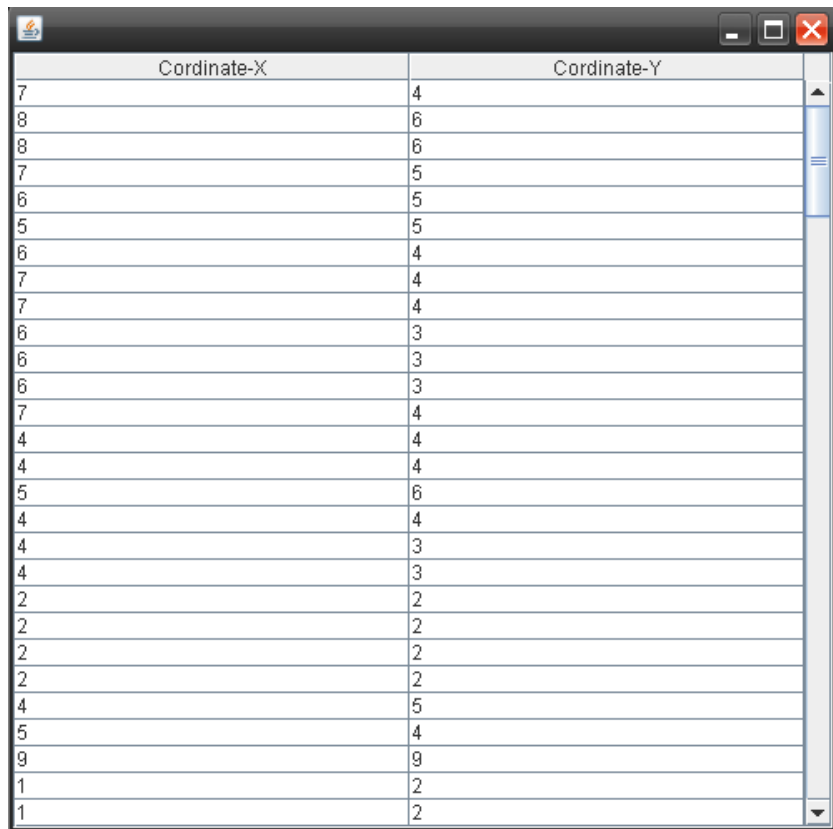


Fig 14

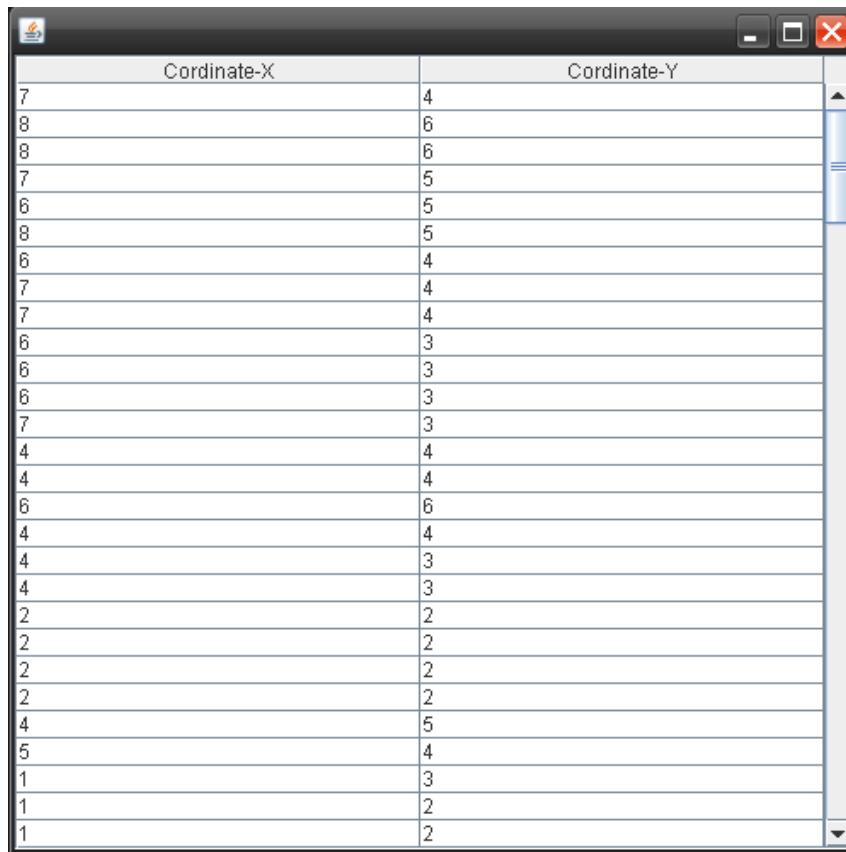
Coordinate-X	Coordinate-Y
7	5
8	6
6	5
5	4
7	4
4	4
2	2
3	4
4	5
3	4
3	4
3	5
1	2
2	5
8	3
6	5
5	4
2	4
1	5
4	5
1	2
6	3
5	4
3	4
6	5
2	5
3	4
4	4

Fig 15



Coordinate-X	Coordinate-Y
7	4
8	6
8	6
7	5
6	5
5	5
6	4
7	4
7	4
6	3
6	3
6	3
7	4
4	4
4	4
5	6
4	4
4	3
4	3
2	2
2	2
2	2
2	2
4	5
5	4
9	9
1	2
1	2

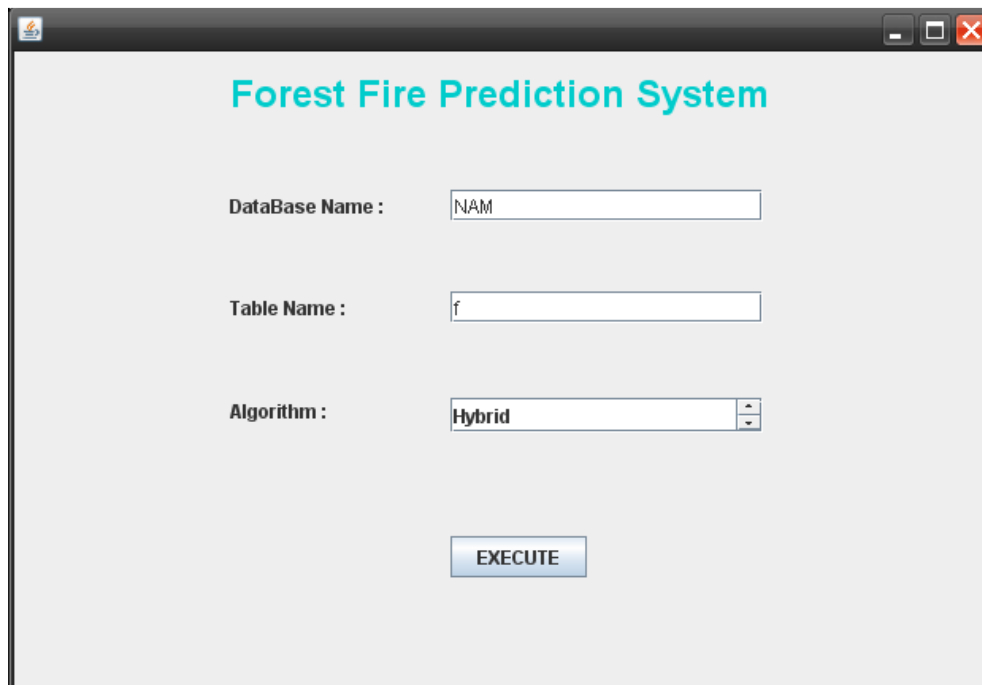
Fig 16



Coordinate-X	Coordinate-Y
7	4
8	6
8	6
7	5
6	5
8	5
6	4
7	4
7	4
6	3
6	3
6	3
7	3
4	4
4	4
6	6
4	4
4	3
4	3
2	2
2	2
2	2
2	2
2	2
4	5
5	4
1	3
1	2
1	2

Fig 17

### 5.9.2 Snapshots With Hybrid Selected



**Forest Fire Prediction System**

DataBase Name :

Table Name :

Algorithm :

Fig 18

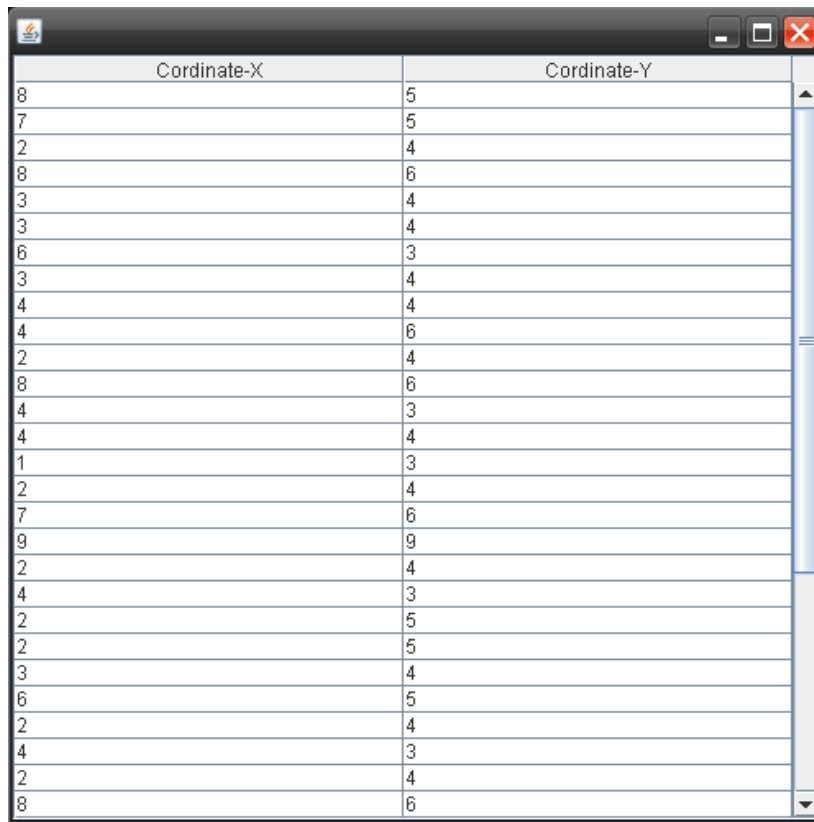


Fig 19

Coordinate-X	Coordinate-Y
6	5
1	2
8	6
7	4
8	6
1	3
7	5
2	5
8	6
1	5
9	4
9	5
3	4
4	5
1	2
8	6
1	3
9	5
4	5
4	3
2	4
4	4
2	2
9	4
2	5
4	3
7	5
1	5

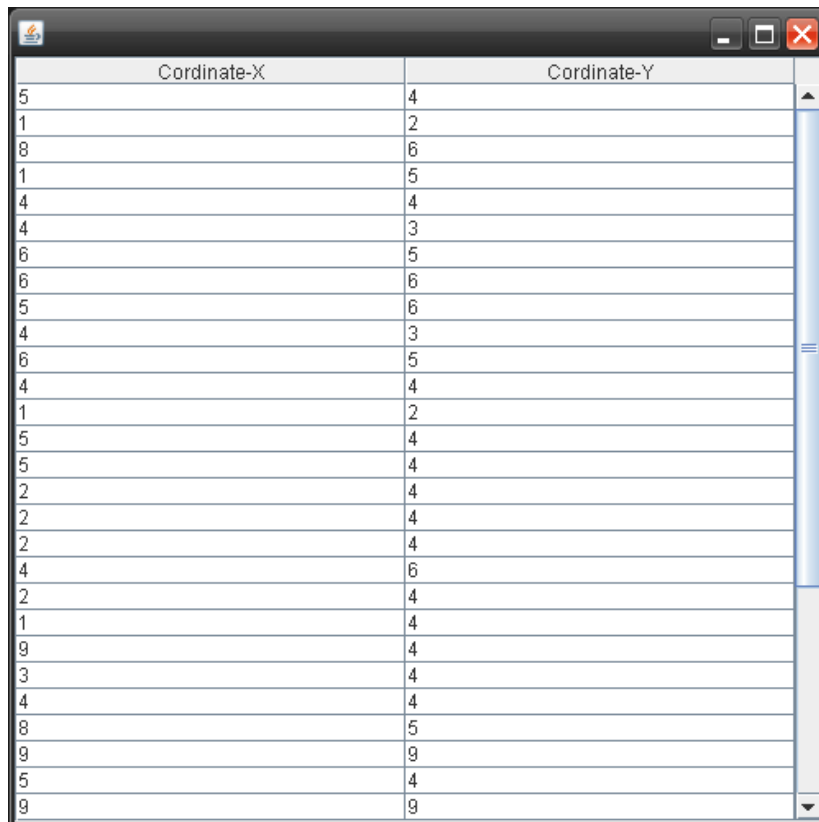
Fig 20





Cordinate-X	Cordinate-Y
8	5
7	5
2	4
8	6
3	4
3	4
6	3
3	4
4	4
4	6
2	4
8	6
4	3
4	4
1	3
2	4
7	6
9	9
2	4
4	3
2	5
2	5
3	4
6	5
2	4
4	3
2	4
8	6

Fig 21



Coordinate-X	Coordinate-Y
5	4
1	2
8	6
1	5
4	4
4	3
6	5
6	6
5	6
4	3
6	5
4	4
1	2
5	4
5	4
2	4
2	4
2	4
4	6
2	4
1	4
9	4
3	4
4	4
8	5
9	9
5	4
9	9

Fig 22

### 5.9.3 Snapshots Of The Dataset Used

Forest Fires													
Temp	Rh	Name	Id	Month	FFMC	DMC	DC	ISI	X	Y	Wind	Rain	Area
8.2	51	fri	1	mar	86.2	26.2	94.3	5.1	7	5	6.7	0	0
18	33	tue	2	oct	90.6	35.4	669.1	6.7	7	4	0.9	0	0
14.6	33	sat	3	oct	90.6	43.7	686.9	6.7	7	4	1.3	0	0
8.3	97	fri	4	mar	91.7	33.3	77.5	9	8	6	4	0.2	0
11.4	99	sun	5	mar	89.3	51.3	102.2	9.6	8	6	1.8	0	0
22.2	29	sun	6	aug	92.3	85.3	488	14.7	8	6	5.4	0	0
24.1	27	mon	7	aug	92.3	88.9	495.6	8.5	8	6	3.1	0	0
8	86	mon	8	aug	91.5	145.4	608.2	10.7	8	6	2.2	0	0
13.1	63	tue	9	sep	91	129.5	692.6	7	8	6	5.4	0	0

Fig. 23

1. X - x-axis spatial coordinate
2. Y - y-axis spatial coordinate
3. Month - month of the year: 'jan' to 'dec'
4. Day - day of the week: 'mon' to 'sun'
5. FFMC – Fine Fuel Moisture Code index from the Forest Fire Weather Index (FWI) System
6. DMC – Duff Moisture Code index from the FWI system
7. DC – Drought Code index from the FWI system
8. ISI – Initial Spread Index from the FWI system

9. Temp - temperature in Celsius degrees
10. RH - relative humidity in & percent
11. Wind - wind speed in km/h
12. Rain - outside rain in mm/m<sup>2</sup>
13. Area - the burned area of the forest (in ha)

## **CONCLUSION**

Successfully Developed Graphical User Interface for Forest Fire Predication System along with the implementation of K-Means and Genetic K-Means Algorithm where GKA helped improve the predictability of forest fires, since K-Means does clustering on entire database and the Genetic Algorithm selects only the fittest and most probable solution for occurrence of forest fires. Genetic K-Means algorithm provides the globally optimum result for the forest fire prediction system.

## BIBLIOGRAPHY

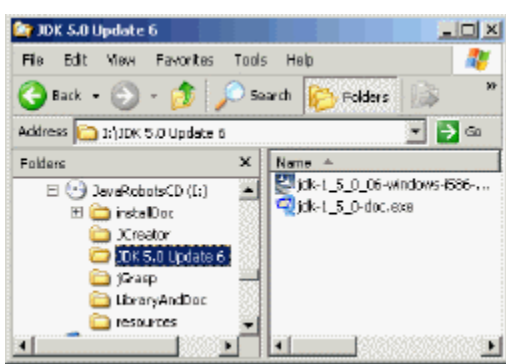
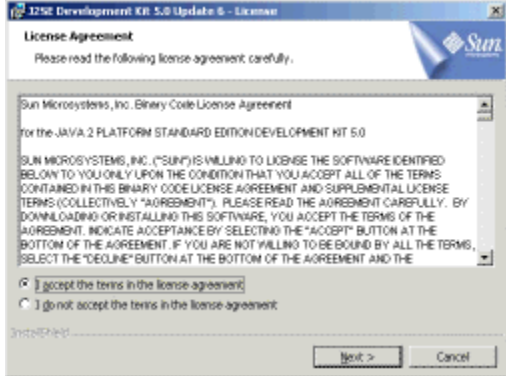
- Novel Hybrid Hierarchical-K-means Clustering Method (H-K-means) for Microarray Analysis by Bernard Chen, Phang C. Tai, R. Harrison and Yi Pan
- Swarm Intelligence Algorithms for Data Clustering by Ajith Abraham, Swagatam Das, and Sandip Roy
- Data Mining by A.K.Pujari
- The KDD process for Extracting Useful Knowledge From Volumes Of Data, Usama Fayyad, Gregory Piatetsky –Shapiro and Padhraic Smyth 1996.
- A Data Mining Approach to predict Forest Fires Using Meteorological Data, Paulo Cortez and Anibal Morais
- Genetic K-Means Algorithm, K. Krishna and M. Narasimha Murty, 1999
- [Http://www.wikipedia.com](http://www.wikipedia.com)
- [Http://www.google.co.in](http://www.google.co.in)
- [Http://www.dsi.uminho.pt/~pcortez](http://www.dsi.uminho.pt/~pcortez)
- [Http://www.machinelearningrepository.com](http://www.machinelearningrepository.com)
- [Http://www-2.cs.cmu.edu/~awm/tutorial/kmeans.html](http://www-2.cs.cmu.edu/~awm/tutorial/kmeans.html)
- [Http://fconyx.ncifcrf.gov/~lukeb/kmeans.html](http://fconyx.ncifcrf.gov/~lukeb/kmeans.html)
- [Http://www.obitko.com/tutorials/genetic-algorithms.html](http://www.obitko.com/tutorials/genetic-algorithms.html)

## APPENDIX

### I. How To Install JDK?

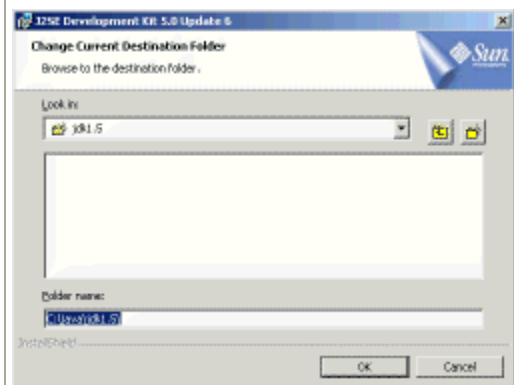
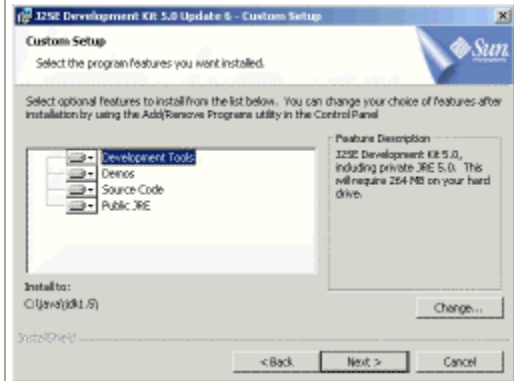
The Java Development Kit or JDK is the software that allows us to develop and run Java programs. Click the image if you need to see a full-sized version.

The following steps will create a directory (folder) on your disk where all your Java-related software will be placed. In the following steps, we will assume a directory on your C: drive named java. If you choose a different place or name, you must be *very* careful to use your name wherever you see C:\java throughout the installation.

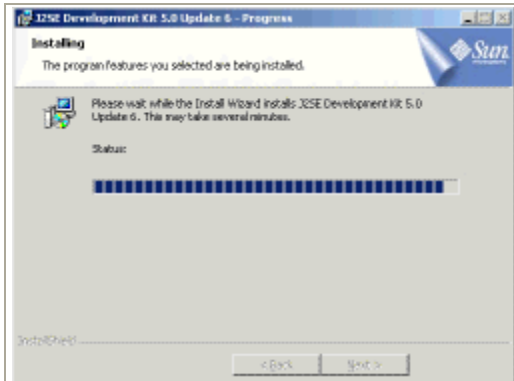
1.	<p>Find the JDK installation files on the CD. They're inside a directory (folder) named JDK 5.0 Update 6. The file has the name jdk-1_5_0_06-windows-i586-p.exe.</p> <p>Remember that you can click on each of the images on this page to see a full-size version.</p>	
2.	<p>Double-click the file to begin installation. A splash screen will show for a while and then you'll be shown a licence agreement. If you want to learn Java, you must click <i>I accept</i> and the <i>Next</i> button.</p>	

This screen allows you to customize your installation.

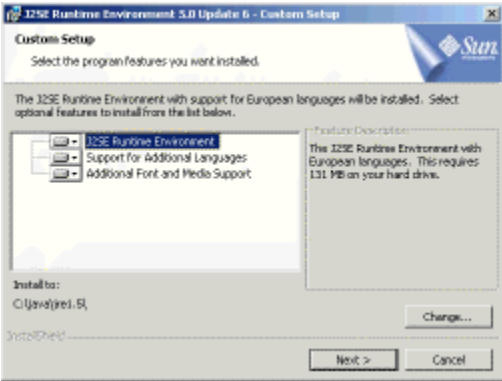
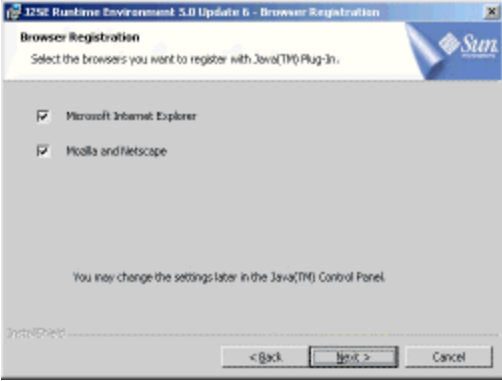
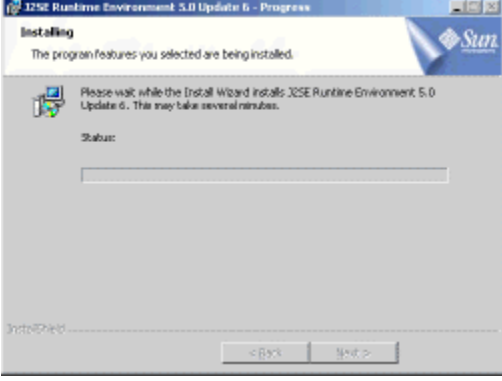
- The defaults shown in the upper part of the dialog box are all OK.
- Change the installation directory so that all of your Java-related programs are together, as follows:
  - Click on the *Change* button.
  - In the resulting screen, type `C:\java\jdk1.5\` in the space for the *Folder name*:
  - Click *OK*.
- You should now see the previous dialog box, shown above. Check the *Install to:* information to verify that it says `C:\java\jdk1.5\`.


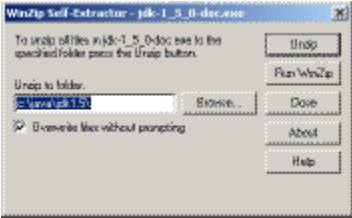



Click *Next*. A progress screen will show for quite a while. It may take several minutes to fully install the software.



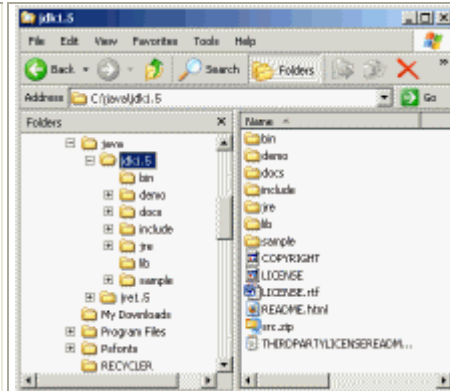


5.	<p>You will be asked to install the Java Runtime Environment. Notice that the screen is almost identical to the one in step 3. As we did there, accept the default settings except for the installation directory. Click the <i>Change...</i> button and change the <i>Folder name:</i> field to C:\java\jre1.5\.</p> <p>Note that this name contains <b>jre</b> whereas the name entered in step 3 contains <b>jdk</b>.</p> <p>Click <i>OK</i> to come back to this screen. Then click <i>Next</i>.</p>	
6.	<p>This is your opportunity to register your Web browser with the new version of Java, enabling applets to run with the new software. Leave all of the browsers listed selected, then click <i>Next</i>.</p>	
7.	<p>You will again be shown a progress screen while software is installed.</p>	

8.	Eventually, you will be shown a completion screen. Click <i>Finish</i> .	
9.	Installing the documentation for the JDK is, strictly speaking, optional. You'll be grateful for it in the later part of the book, so you may as well do it now.  Go back to the CD and find the file jdk-1_5_0-doc.exe inside the directory named JDK 5.0 Update 6. Double-click on it to open it.	
10.	It should already have C:\java\jdk1.5\ filled in the text field labeled <i>Unzip to folder</i> . If you chose a different place to store the JDK, type it in or use the <i>Browse</i> button to go find it. Then click the <i>Unzip</i> button. A progress bar will appear at the bottom. This may take several minutes.  When the completion message is shown, click <i>OK</i> and then <i>Close</i> on the dialog shown above.	

11.

You're done installing the JDK! Just as a double-check, take a look at the directory where you installed it, C:\java\jdk1.5\ . It should look like the one shown here.



## II. How To Install JDBC

Java Database Connectivity (**JDBC**) is a programming framework for Java developers writing programs that access information stored in databases, spreadsheets, and flat files. JDBC is commonly used to connect a user program to a "behind the scenes" database, regardless of what database management software is used to control the database. In this way, JDBC is cross-platform. This article will provide an introduction and sample code that demonstrates database access from Java programs that use the classes of the JDBC API, which is available for free download from Sun's site.

A database that another program links to is called a data source. Many data sources, including products produced by Microsoft and Oracle, already use a standard called Open Database Connectivity (ODBC). Many legacy C and Perl programs use ODBC to connect to data sources. ODBC consolidated much of the commonality between database management systems. JDBC builds on this feature, and increases the level of abstraction. JDBC-ODBC bridges have been created to allow Java programs to connect to ODBC-enabled database software.

This article assumes that readers already have a data source established and are moderately familiar with the Structured Query Language (SQL), the command language for adding records, retrieving records, and other basic database manipulations. See Hoffman's tutorial on SQL if you are a beginner or need some refreshing.

### ***Using a JDBC driver***

Regardless of data source location, platform, or driver (Oracle, Microsoft, etc.), JDBC makes connecting to a data source less difficult by providing a collection of classes that abstract details of the database interaction. Software engineering with JDBC is also conducive to module reuse. Programs can easily be ported to a different infrastructure for which you have data stored (whatever platform you choose to use in the future) with only a driver substitution.

As long as you stick with the more popular database platforms (Oracle, Informix, Microsoft, MySQL, etc.), there is almost certainly a JDBC driver written to let your programs connect and manipulate data. You can download a specific JDBC driver from the manufacturer of your database management system (DBMS) or from a third party (in the case of less popular open source products) [5]. The JDBC driver for your database will come with specific instructions to make the class files of the driver available to the Java Virtual Machine, which your program is going to run. JDBC drivers use Java's built-in DriverManager to open and access a database from within your Java program.

To begin connecting to a data source, you first need to instantiate an object of your JDBC driver. This essentially requires only one line of code, a command to the DriverManager, telling the Java Virtual Machine to load the bytecode of your driver into memory, where its methods will be available to your program. The String parameter below is the fully qualified class name of the driver you are using for your platform combination:

```
Class.forName("org.gjt.mm.mysql.Driver").newInstance();
```

### ***Connecting to your database***

To actually manipulate your database, you need to get an object of the Connection class from your driver. At the very least, your driver will need a URL for the database and parameters for access control, which usually involves standard password authentication for a database account.

As you may already be aware, the Uniform Resource Locator (URL) standard is good for much more than telling your browser where to find a web page:

`http://www.vusports.com/index.html`

The URL for our example driver and database looks like this:

`jdbc:mysql://db_server:3306/contacts/`

Even though these two URLs look different, they are actually the same in form: the protocol for connection, machine host name and optional port number, and the relative path of the resource. Your JDBC driver will come with instructions detailing how to form the URL for your database. It will look similar to our example.

You will want to control access to your data, unless security is not an issue. The standard least common denominator for authentication to a database is a pair of strings, an account and a password. The account name and password you give the driver should have meaning within your DBMS, where permissions should have been established to govern access privileges.

Our example JDBC driver uses an object of the Properties class to pass information through the DriverManager, which yields a Connection object:

```
Properties props = new Properties();
props.setProperty("user", "contacts");
props.setProperty("password", "blackbook");
Connection con = DriverManager.getConnection(
    "jdbc:mysql://localhost:3306/contacts/", props);
```

Now that we have a Connection object, we can easily pass commands through it to the database, taking advantage of the abstraction layers provided by JDBC.

### ***Structuring statements***

Databases are composed of tables, which in turn are composed of rows. Each database table has a set of rows that define what data types are in each record. Records are also stored as rows of the database table with one row per record. We use the data source connection created in the last section to execute a command to the database.

We write commands to be executed by the DBMS on a database using SQL. The syntax of a SQL statement, or query, usually consists of an action keyword, a target table name, and some parameters. For example:

```
INSERT INTO songs VALUES (  
    "Jesus Jones", "Right Here, Right Now");  
INSERT INTO songs VALUES (  
    "Def Leppard", "Hysteria");
```

These SQL queries each added a row of data to table "songs" in the database. Naturally, the order of the values being inserted into the table must match the order of the corresponding columns of the table, and the data types of the new values must match the data types of the corresponding columns. For more information about the supported data types in your DBMS, consult your reference material.

To execute an SQL statement using a Connection object, you first need to create a Statement object, which will execute the query contained in a String.

```
Statement stmt = con.createStatement();  
String query = ... // define query  
stmt.executeQuery(query);
```