**CHAPTER NUMBER 1**

**INTRODUCTION**

* 1. **Background**

The art of concealing messages is known as Steganography. It has been in use since ancient times wherein secret messages where concealed in pictures which could be traced by viewing it through different angles. After the advent of computers, there has been a drastic increase in image steganography which is oblivious to the human eyes. Here we’ll explore a a novel method of compression that revolves around JPEG compression.

* 1. **Problem Statement**

JPEG compression revolves around DCT transformation and quantization of matrix coefficients. The quantization matrix is such that most of the coefficients of the matrix are zero. This reduces the amount of information that could be embedded in the image significantly affecting communication. Here we try and solve this problem with a change in the quantization matrix.

* 1. **Importance**

JPEG-JSTEG is a robust method of steganography that survives quite a few image manipulations aiming at corrupting data of images. Increasing the amount of information that can be embedded into the image would greatly enhance steganography. A lot of information could be transmitted in a sublime channel in a very robust manner. Although it would reduce the amount of compression that could be carried out in a JPEG image, it can be disregarded due to the merits over the pits of this technique.

* 1. **Organization of report**

The rest of the report is organized as follows. In section “Introduction” we give a brief description of the compression process and the problem statement. In Section “Overview and Planning” we introduce our assumptions, challenges project schedule etc. In Section “Literature Survey and Review” we provide a brief Literature survey. In section “System Design” we elaborate upon our algorithm. In section ”System Implementation” we provide our sample code implementation. In section “Results and Discussion” we highlight our results and also analyze them. Finally in section “Conclusion” we conclude our project.

**CHAPTER NUMBER 2**

**OVERVIEW and PLANNING**

* 1. **Assumptions**
     1. **Evaluation Criteria**

To access our technique we use the size of both the files computed with different quantization tables. We then compare the information stored in both and decide the practicality of out technique.

* + 1. **Adversary Model**

In this project the model proposed by Chin-Chen Chang, Tung-Shou Chen, Lou-Zo Chung is implemented and its advantages and disadvantages compared with that of JPEG-JSTEG. The observations are then noted.

* 1. **Challenges**

The following are the challenges involved in the project:-

1. Improve the quality of Steganography.
2. Minimize loss in image quality
3. Increase the number of bits to be embedded in
   1. **Software requirements**

The Software requirement for the project is Octave with pakages control, signal, miscellaneous and image. C++ with octave-dev would also be necessary for

* 1. **Project Schedule**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Research Paper |  |  |  |  |
| Implementation |  |  |  |  |
| JPEG-compression |  |  |  |  |
| Huffman Encoding |  |  |  |  |
| Run Length Entropy Encoding |  |  |  |  |
| Comparing both models |  |  |  |  |
| Error checking |  |  |  |  |
| Testing |  |  |  |  |
|  | January | February | March | April |

Table 1: Project Schedule

**CHAPTER NUMBER 3**

**LITERATURE SURVEY and REVIEW**

* 1. **Literature Survey**

We come to know about JPEG images and their compression from Gregory K. Wallace [1]. The suggested approach leads us to FDCT and DCT which can be understood from N Ahmed, T Natarajan, KR Rao [2].

The quantization table and rounding off are simple mathematical concepts whcih are concise and clear to understand. The Huffman Encoding for image steganography is detailed upon in A Nag, S Biswas, D Sarkar, PP Sarkar[4]. The Run Length Encoding can be found in detail when referring to W Chen, MC Lee.

**CHAPTER NUMBER 4**

**SYSTEM DESIGN**

* 1. **Proposed Scheme**

In this project paper, the JPEG-JSTEG method has been challenged with a better method that allows more bits of a string to be embedded into an image while decreasing the compression ratio. We will now look into the complete process of compression of the image and embedding information into the image from reading the image to saving it.

* + 1. **Initialization**

The image is read in MATLAB using imread function. This converts the image into a matrix with dimensions same as that of the image plus an added 3rd dimension that stores the color values for RGB. Every element represents the pixel intensity with a value ranging from 0 to 255.

The image is then converted into 8x8 matrix form which when implemented in octave results in a five dimensional matrix. The matrix is then moved to a DCT or FDCT transform depending upon the choice of implementation. Practically this should not make much of a difference in the implementation.

* + 1. **DCT**

We then take each 8x8 matrix from our five dimensional matrix and then carry out a DCT transformation upon each of them

* + 1. **Rough Intuitionistic Fuzzy C Means**

Going into minor details of Intuitionistic Fuzzy C Means will better explain RIFCM. Conventional Fuzzy C Means cluster the feature vectors by searching for local minima of the objective function

where is the Euclidean distance measure between *vi* (cluster centre) of each region and *xk* (points in the pattern) and *uik* is the membership value of kth data in ith cluster. *c* is the number of clusters, *n* is the number of data points. Minimization of *Jm* is based on suitable selection of U (membership matrix) and *v* using an iterative process:

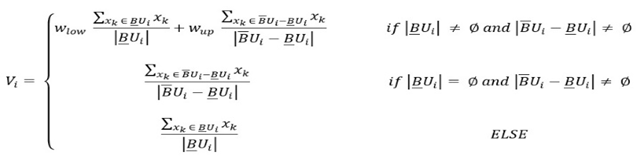
There’s an added factor called the hesitation degree. This is defined as follows:

This hesitation degree adds another parameter that deals with vagueness of data better. Its been proven that IFCM works better than algorithms like FCM and KCM. Now, the updated membership calculation is , where denotes the intuitionistic fuzzy membership matrix and *vi* is updated the same way as above. The final cost function is as follows:

with m = 2

and where where is the hesitation degree of *k*th element in cluster ‘i’.

A rough set, first described by a Polish computer scientist Zdzisław I. Pawlak [8] , is a formal approximation of a crisp set i.e., conventional set in terms of a pair of sets which give the lower and the upper approximation of the original set. In the standard version of rough set theory Pawlak 1991, the lower- and upper-approximation sets are crisp sets, but in other variations, the approximating sets may be fuzzy sets. The lower approximation is a complete set of objects which definitely belong to the solution set. The upper approximation includes all objects which could possibly belong to the solution set. Those objects which are present in the upper approximation but not present in the lower approximation constitute the Boundary region.



The above is the cluster update formula for Rough Sets based algorithms and applies to RIFCM as well.

Given the above theory, Firefly algorithm returns the optimal initial centroids and fuzzy matrix. These are then passed over to RIFCM which is a combination of the above two concepts i.e Rough Sets and Intuitionistic Fuzzy C Means. The performance measures like DB and D index will measure the clustering quality of the algorithm used. With that we can automate the result process and check experimentally whether RIFCM has a performance improvement and gets stabilized.

So the above architecture talks about the algorithm in detail and describes the process of hybridizing Firefly with RFCM and RIFCM.

* + 1. **DB and D index measures**

In order to mathematically compare the clustering quality of the algorithms, two indices have been used: DB index and Dunn index.





Lower the value of DB index and higher the value of D index, the better the clustering quality. RFCM, RFCMFA, RIFCM and RIFCMFA have been tested on three images: Brain MRI, Rice Copy and Satellite images. Each of these four algorithms was run 50 times for each image and the results have been presented in the form of graphs.

For all the images, two cluster centers were considered. The X-axis of each graph represents the iterations and the Y-axis represents the DB/Dunn index values. The maximum and minimum index values are explicitly mentioned for each image and the performance of each algorithm is discussed and visualized.

**CHAPTER NUMBER 5**

**SYSTEM IMPLEMENTATION**

* 1. **Initialization Code**

clear all; close all; clc;

[img] = imread('lena.jpg');

img = double(img);

clusterNum = 3;

for iter1 = 1:1

[Upre, center] = firefly\_own(img,clusterNum);

[ Unow, center, now\_obj\_fcn ] = ifcm2( img, clusterNum , Upre, center);

end

figure;

subplot(3,2,1); imshow(img,[]);

for i=1:clusterNum

subplot(3,2,i+1);

imshow(Unow(:,:,i),[]);

end

* 1. **Firefly algorithm**

function [Upre, center] = firefly\_own(img,clusterNum)

%generating random membership matrices

expoNum = 2;

nof = 15;

[row,col] = size(img);

for x=1:nof

center = 255\* rand(clusterNum,1);

center = sort(center);

ffc(:,:,x) = center;

% for i=1:row

% for j=1:col

for uII = 1:clusterNum

tmp = zeros(row, col);

for uJJ = 1:clusterNum

disUp = abs(img - ffc(uII,:,x));

disDn = abs(img - ffc(uJJ,:,x));

tmp = tmp + ((disUp./disDn).^(2/(expoNum-1)));

end

Uold(:,:, uII) = 1./(tmp);

% Unow(i,j,uII) = 1-(1-Uold(i,j,uII).^alpha).^(1/alpha);

ffm(:,:,uII,x) = Uold(:,:,uII);

end

% end

end

%firefly variables are pre-defined from previous experimental results

beta0 = 1;

gamma = 1.0;

alpha = 0.2;

delta = 0.97;

%code for firefly movement

max\_iter = 50;

for ii =1:nof

I(ii) = 0;

for j= 1:clusterNum

I(ii) = I(ii) + sum(sum((ffm(:,:,j,ii).^2).\*(((img - ffc(j,:,ii)).^2))));

end

I(ii) = 1/(I(ii) + 1);

end

for zzz=1:max\_iter

[I,index] = sort(I);

ffm = ffm(:,:,:,index);

ffc = ffc(:,:,index);

prevI = I;

prevffc = ffc;

prevffm = ffm;

for ii =1:nof

for jj=1:nof

if I(ii) < I(jj)

r = sqrt(sum((ffc(:,:,ii) - ffc(:,:,jj)).^2))/50;

beta = beta0 \* exp(-gamma\*r\*r)

%beta = 0.5;

%beta = 0.75;

ffc(:,:,ii)=ffc(:,:,ii).\*(1-beta)+ffc(:,:,jj).\*beta+alpha.\*(rand-0.5);

% alpha.\*(rand-0.5)

%for i=1:row

% for j=1:col

for uII = 1:clusterNum

tmp = zeros(row, col);

for uJJ = 1:clusterNum

disUp = abs(img - ffc(uII,:,ii));

disDn = abs(img - ffc(uJJ,:,ii));

tmp = tmp + (disUp./disDn).^(2/(expoNum-1));

end

Uold(:,:, uII) = 1./(tmp);

% Unow(i,j,uII) = 1-(1-Uold(i,j,uII).^alpha).^(1/alpha);

ffm(:,:,uII,ii) = Uold(:,:,uII);

end

% end

% end

for j= 1:clusterNum

I(ii) = I(ii) + sum(sum((ffm(:,:,j,ii).^2).\*(((img - ffc(j,:,ii)).^2))));

end

I(ii) = 1/(I(ii) + 1);

end

if I(ii) < prevI(ii)

I(ii) = prevI(ii);

ffc(:,:,ii) = prevffc(:,:,ii);

ffm(:,:,:,ii) = prevffm(:,:,:,ii);

end

end

end

alpha = alpha .\* delta;

end

Upre = ffm(:,:,:,nof);

center = ffc(:,:,nof)

* 1. **RFCM code**

function [ Upre, center, now\_obj\_fcn ] = RFCM( img, clusterNum,Upre,center )

if nargin < 2

clusterNum = 2; % number of cluster

end

%center = rand(clusterNum,1).\*255;

alpha = 2;

[row, col] = size(img);

expoNum = 2; % fuzzification parameter

epsilon = 0.001; % stopping condition

mat\_iter = 25; % number of maximun iteration

mean\_data = sum(sum(img))/(row\*col);

variance\_data = sum(sum(abs((img - mean\_data))))/(row\*col);

pre\_obj\_fcn = 0;

for i=1:clusterNum

pre\_obj\_fcn = pre\_obj\_fcn + sum(sum((Upre(:,:,i) .\*((img - center(i)).^2))));

end

fprintf('Initial objective fcn = %f\n', pre\_obj\_fcn);

%U1low = zeros(size(img));

%U2low = zeros(size(img));

Ulow = zeros([size(img), clusterNum]);

for iter = 1:mat\_iter

center

Unow = zeros(size(Upre));

%for i=1:row

%for j=1:col

for uII = 1:clusterNum

tmp = zeros(row,col);

for uJJ = 1:clusterNum

%disUp = abs(img - center(uII));

%disDn = abs(img - center(uJJ));

%kernel part

disUp = 2 \* (1 - exp(-abs(img - center(uII))/variance\_data));

disDn = 2 \* (1 - exp(-abs(img - center(uJJ))/variance\_data));

tmp = tmp + ((disUp./disDn).^(2/(expoNum-1)));

end

Uold(:,:, uII) = 1./(tmp);

%Unow(i,j,uII) = 1-(1-Uold(i,j,uII).^alpha).^(1/alpha);

end

%end

%end

for i = 1:row

for j = 1:col

for k = 1:clusterNum

if Ulow(i,j,k) ~= 0

Uold(i,j,k) = 1;

end

end

end

end

Unow = Uold;

now\_obj\_fcn = 0;

Upre=Unow;

Unow = Unow.^expoNum;

for i=1:clusterNum

now\_obj\_fcn = now\_obj\_fcn + sum(sum((Unow(:,:,i) .\*((img - center(i)).^2))));

end

fprintf('Iter = %d, Objective = %f\n', iter, now\_obj\_fcn);

% Adding the Rough Part

Ulow = zeros([size(img),clusterNum]);

Uup = Ulow;

DB\_Ulow = Ulow;

DB\_Uup = Ulow;

num\_elts\_low = zeros(clusterNum);

num\_elts\_up = zeros(clusterNum);

pair\_value = zeros(size(img));

x4=0; %debug variable

%Urough = Upre(:,:,1) - Upre(:,:,2);

[temp, index] = sort(Upre,3);

for i=1:row

for j=1:col

Urough(i,j) = temp(i,j,clusterNum) - temp(i,j,clusterNum-1);

pair\_value(i,j) = index(i,j,clusterNum)\*10 + index(i,j,clusterNum-1);

end

end

%vectorized implementation of the above

%Urough = temp(:,:,clusterNum) - temp(:,:,clusterNum-1);

%pair\_value = index(:,:,clusterNum)\*10 + index(:,:,clusterNum-1);

thold = sum(sum(abs(Urough)))/(row\*col);

for i = 1:row

for j=1:col

if Urough(i,j) < thold

%U1up(i,j) = img(i,j);

%U2up(i,j) = img(i,j);

%x3 = x3 + 1;

up\_c\_no = [mod(pair\_value(i,j),10) floor(pair\_value(i,j)/10)];

Uup(i,j,up\_c\_no(1)) = img(i,j);

DB\_Uup(i,j,up\_c\_no(1)) = (img(i,j) - center(up\_c\_no(1),1)).^2;

Uup(i,j,up\_c\_no(2)) = img(i,j);

DB\_Uup(i,j,up\_c\_no(2)) = (img(i,j) - center(up\_c\_no(2),1)).^2;

num\_elts\_up(up\_c\_no(1)) = num\_elts\_up(up\_c\_no(1)) + 1;

num\_elts\_up(up\_c\_no(2)) = num\_elts\_up(up\_c\_no(2)) + 1;

elseif Urough(i,j) >= thold

%U1low(i,j) = img(i,j);

%x1 = x1 + 1;

low\_c\_no = floor(pair\_value(i,j)/10);

Ulow(i,j,low\_c\_no) = img(i,j);

DB\_Ulow(i,j,low\_c\_no) = (img(i,j) - center(low\_c\_no,1)).^2;

num\_elts\_low(low\_c\_no) = num\_elts\_low(low\_c\_no) + 1;

end

end

end

if max(max(max(abs(Unow-Upre))))<epsilon || abs(now\_obj\_fcn - pre\_obj\_fcn)<epsilon

break;

else

%Upre = Unow.^expoNum;

for k=1:clusterNum

if num\_elts\_low(k) ~= 0 && num\_elts\_up(k) ~= 0

center(k,1) = (0.9 \* sum(sum( Ulow(:,:,k) )) / num\_elts\_low(k)) + (0.1 \* sum(sum( Unow(:,:,k).\*Uup(:,:,k) )) / (sum(sum( Unow(:,:,k) ))));

elseif num\_elts\_low(k) == 0 && num\_elts\_up(k) ~= 0

center(k,1) = sum(sum( Unow(:,:,k).\*Uup(:,:,k) )) / (sum(sum( Unow(:,:,k) )));

else

center(k,1) = sum(sum( Ulow(:,:,k) ))/ num\_elts\_low(k);

end

end

pre\_obj\_fcn = now\_obj\_fcn;

end

end

%Adding DB Index Code

s=zeros(clusterNum,1);

for k=1:clusterNum

if num\_elts\_low(k) ~= 0 && num\_elts\_up(k) ~= 0

s(k,1) = (0.9 \* sum(sum( DB\_Ulow(:,:,k) )) / num\_elts\_low(k)) + (0.1 \* sum(sum( Unow(:,:,k).\*DB\_Uup(:,:,k) )) / (sum(sum( Unow(:,:,k) ))));

elseif num\_elts\_low(k) == 0 && num\_elts\_up(k) ~= 0

s(k,1) = sum(sum( Unow(:,:,k).\*DB\_Uup(:,:,k) )) / (sum(sum( Unow(:,:,k) )));

else

s(k,1) = sum(sum( DB\_Ulow(:,:,k) ))/ num\_elts\_low(k);

end

end

temp\_1 = 0;

for i=1:clusterNum

for j=1:clusterNum

if(i == j)

arr(j) = 0;

else

arr(j) = (s(i,1) + s(j,1))/abs((center(i,1) - center(j,1)));

end

end

temp = max(arr);

temp\_1 = temp\_1 + temp;

end

db\_index = (1/clusterNum)\*temp\_1

%Adding Dunn Index Code

for i=1:clusterNum

for j=1:clusterNum

if(i == j)

arr(j) = 100000;

else

arr(j) = abs(center(i,1) - center(j,1));

end

end

temp(i) = min(arr);

end

dunn = min(temp)/max(s)

* 1. **RIFCM code**

function [ Upre, center, now\_obj\_fcn ] = RIFCM( img, clusterNum,Upre,center )

% demo

% img = double(imread('brain.tif'));

% clusterNum = 3;

% [ Unow, center, now\_obj\_fcn ] = FCMforImage( img, clusterNum );

% figure;

% subplot(2,2,1); imshow(img,[]);

% for i=1:clusterNum

% subplot(2,2,i+1);

% imshow(Unow(:,:,i),[]);

% end

%debug

if nargin < 2

clusterNum = 2; % number of cluster

end

center = rand(2,1).\*255

alpha = 2;

[row, col] = size(img);

expoNum = 2; % fuzzification parameter

epsilon = 0.001; % stopping condition

mat\_iter = 1250; % number of maximun iteration

pre\_obj\_fcn = 0;

for i=1:clusterNum

pre\_obj\_fcn = pre\_obj\_fcn + sum(sum((Upre(:,:,i) .\*((img - center(i)).^2))));

end

fprintf('Initial objective fcn = %f\n', pre\_obj\_fcn);

U1low = zeros(size(img));

U2low = zeros(size(img));

for iter = 1:mat\_iter

center

Unow = zeros(size(Upre));

%for i=1:row

%for j=1:col

for uII = 1:clusterNum

tmp = zeros(row,col);

for uJJ = 1:clusterNum

disUp = abs(img - center(uII));

disDn = abs(img - center(uJJ));

tmp = tmp + ((disUp./disDn).^(2/(expoNum-1)));

end

Uold(:,:, uII) = 1./(tmp);

%Unow(i,j,uII) = 1-(1-Uold(i,j,uII).^alpha).^(1/alpha);

end

%end

%end

for i = 1:row

for j = 1:col

if U1low(i,j) ~= 0

Uold(i,j,1) = 1;

end

if U2low(i,j) ~= 0

Uold(i,j,2) = 1;

end

end

end

%adding hesitation factor

%for i=1:row

%for j=1:col

for uII = 1:clusterNum

hes(:,:,uII) = 1 - Uold(:,:,uII) - (1-Uold(:,:,uII).^alpha).^(1/alpha);

end

%end

%end

sumx=0;

for uII=1:clusterNum

hes\_star(uII) = (1/(row\*col)).\* sum(sum(hes(:,:,uII)));

sumx = sumx + hes\_star(uII).\*exp(1 - hes\_star(uII));

end

Unow = Uold + hes;

now\_obj\_fcn = 0;

Upre=Unow;

Unow = Unow.^expoNum;

for i=1:clusterNum

now\_obj\_fcn = now\_obj\_fcn + sum(sum((Unow(:,:,i) .\*((img - center(i)).^2))));

end

now\_obj\_fcn = now\_obj\_fcn + sumx;

fprintf('Iter = %d, Objective = %f\n', iter, now\_obj\_fcn);

% Adding the Rough Part

U1low = zeros(size(img));

U1up = zeros(size(img));

U2low = zeros(size(img));

U2up = zeros(size(img));

x1 = 0;

x2 = 0;

x3 = 0;

x4=0; %debug variable

Urough = Upre(:,:,1) - Upre(:,:,2);

thold = sum(sum(abs(Urough)))/(row\*col)

for i = 1:row

for j=1:col

if Urough(i,j) <= -thold

U2low(i,j) = img(i,j);

x2 = x2 + 1;

elseif Urough(i,j) > -thold && Urough(i,j) <= thold

U1up(i,j) = img(i,j);

U2up(i,j) = img(i,j);

x3 = x3 + 1;

elseif Urough(i,j) > thold

U1low(i,j) = img(i,j);

x1 = x1 + 1;

end

end

end

x1

x2

x3

if max(max(max(abs(Unow-Upre))))<epsilon || abs(now\_obj\_fcn - pre\_obj\_fcn)<epsilon

break;

else

%Upre = Unow.^expoNum;

if x1 ~= 0 && (x3) ~= 0

center(1,1) = (0.9 \* sum(sum( U1low )) / x1) + (0.1 \* sum(sum( Unow(:,:,1).\*U1up )) / (sum(sum( Unow(:,:,1) ))));

elseif x1 == 0 && (x3) ~= 0

center(1,1) = sum(sum( Unow(:,:,1).\*U1up )) / (sum(sum( Unow(:,:,1) )));

else

center(1,1) = sum(sum( U1low ))/ x1;

end

if x2 ~= 0 && (x3)~= 0

center(2,1) = (0.9 \* sum(sum( U2low )) / x2) + (0.1 \* sum(sum( Unow(:,:,2).\*U2up )) / (sum(sum( Unow(:,:,2) ))));

elseif x2 == 0 && (x3) ~= 0

center(2,1) = sum(sum( Unow(:,:,2).\*U2up )) / (sum(sum( Unow(:,:,2) )));

else

center(2,1) = sum(sum( U2low ))/ x2;

end

pre\_obj\_fcn = now\_obj\_fcn;

end

end

* 1. **DB and DUNN index**

function [dbi,dunn] = dbindex(Upre,center,img,clusterNum)

[row,col] = size(img);

s=zeros(clusterNum,1);

for i=1:clusterNum

s(i,1) = ((1/(row\*col))\*sum(sum((Upre(:,:,i).\*img - center(i,1)).^2))).^(1/2);

end

m=abs(center(1,1)-center(2,1));

r = (s(1,1) + s(2,1))/m;

dbi = r;

t = max(s);

dunn = m/t;

**CHAPTER NUMBER 6**

**RESULTS and DISCUSSION**

* 1. **Output/Results**

Project paper titled, “**Image Segmentation using Rough Intuitionistic Fuzzy C Means hybridized with Firefly algorithm**” accepted for publication in the Springer book Smart Innovation, Systems and Technologies, <http://www.springer.com/series/8767> , as proceedings of the Smart Systems, Innovations and Computing(SSIC 2017) conference.

Successfully implemented the proposed scheme in Matlab.

* 1. **Result Analysis**

RFCM, RFCMFA, RIFCM and RIFCMFA have been tested on three images: Brain MRI, Rice Copy and Satellite images. Each of these four algorithms was run 50 times for each image and the results have been presented in the form of graphs.

For all the images, two cluster centers were considered. The X-axis of each graph represents the iterations and the Y-axis represents the DB/Dunn index values. The maximum and minimum index values are explicitly mentioned for each image and the performance of each algorithm is discussed and visualized.

**Brain MRI**



Figure 1: The first image is the input image and the following segmented images are the outputs after applying RFCM, RFCMFA, RIFCM, and RIFCMFA respectively

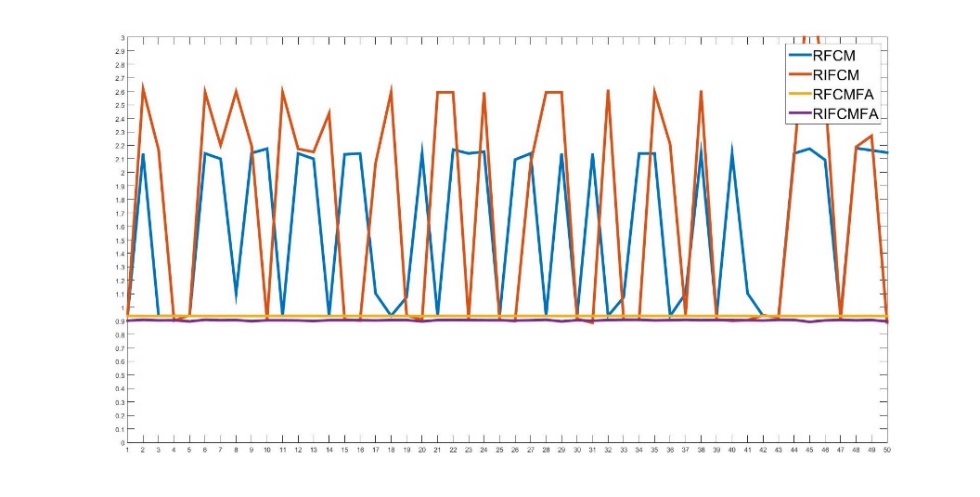


Figure 2: DB values for Brain MRI

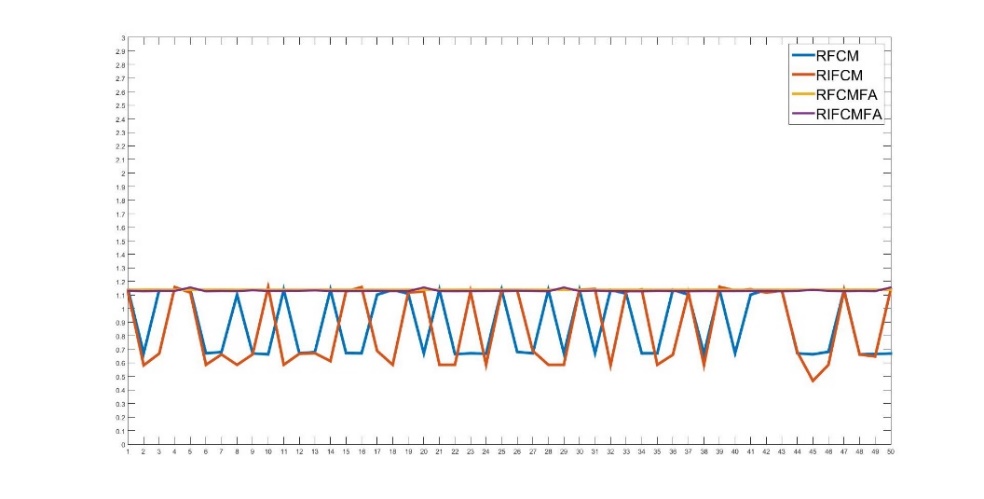


Figure 3: Dunn values for Brain MRI

From the graphs, it is evident that Firefly is stabilizing the performance of both RFCM and RIFCM. The average DB value of RFCM is 1.6, RIFCM is 1.4, RFCMFA is 0.93 and RIFCMFA is 0.91. The Dunn values are also very consistent for RFCMFA and RIFCMFA. Both the hybrid algorithms converged within 7 iterations whereas RIFCM and RFCM took 15 iterations on an average. Hence, it can be clearly observed that Firefly algorithm has eliminated the randomness associated with RFCM and RIFCM.

**Satellite Image**



Figure 4: The first image is the input image and the following segmented images are the outputs after applying RFCM, RFCMFA, RIFCM, and RIFCMFA respectively

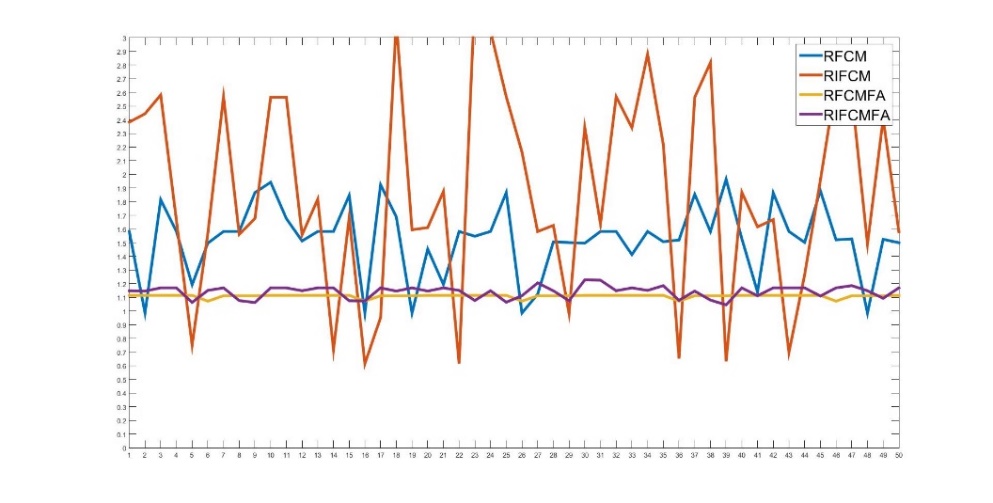


Figure 5: DB values for satellite image

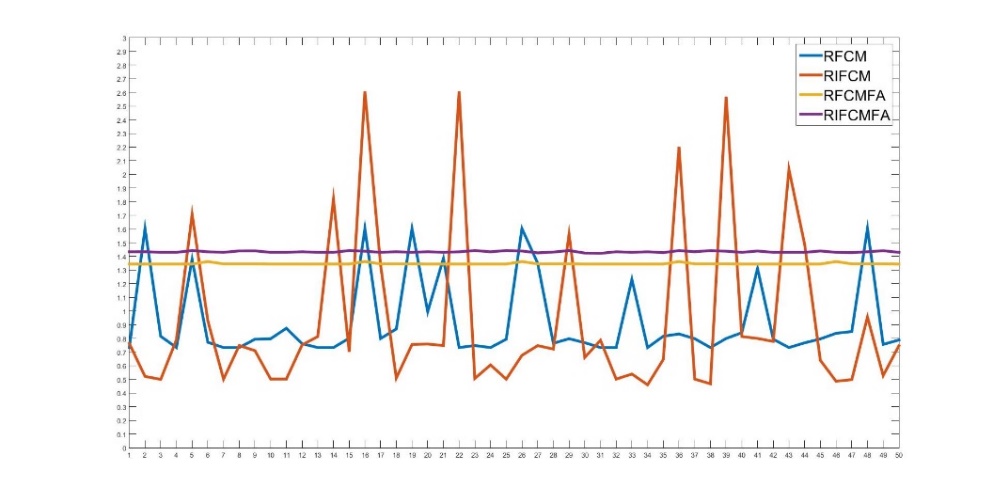


Figure 6: Dunn values for satellite image

Depending upon the initial random cluster centers, the DB and Dunn values of RIFCM and RFCM are either very good or very bad. The average number of iterations for RFCM and RIFCM are 34. For RIFCMFA and RFCMFA, the average number of iterations is 19.

**CHAPTER NUMBER 7**

**CONCLUSION**

* 1. **Conclusion**

In this project, we propose a novel algorithm to stabilize the two rough sets based clustering algorithms: RFCM and RIFCM. Our experimental results prove that using Firefly algorithm to provide the near optimal cluster centroids and fuzzy membership matrix eliminates the susceptibility of RFCM and RIFCM to random initialization of data. This results in a stable and reliable clustering output. Moreover, the number of iterations required for convergence has also reduced considerably. Our future works include analyzing image characteristics such as the optimal number of clusters and trying to use Firefly algorithm for finding the appropriate number of reducts.

**REFERENCES**

1. Atanassov, K. T.: Intuitionistic fuzzy sets. *Fuzzy sets and Systems*, 20(1), 87-96 (1986)
2. Bezdek, J. C., Ehrlich, R., & Full, W.: FCM: The fuzzy c-means clustering algorithm. *Computers & Geosciences*, 10(2-3), 191-203 (1984)
3. Bhargava, R., Tripathy, B. K., Tripathy, A., Dhull, R., Verma, E., & Swarnalatha, P.: Rough intuitionistic fuzzy c-means algorithm and a comparative analysis. In *Proceedings of the 6th ACM India Computing Convention* (p. 23). ACM (2013, August)
4. Chaira, T.: A novel intuitionistic fuzzy C means clustering algorithm and its application to medical images. *Applied Soft Computing*, 11(2), 1711-1717 (2011)
5. Lingras, P., & West, C.: Interval set clustering of web users with rough k-means. *Journal of Intelligent Information Systems*, 23(1), 5-16 (2004)
6. Maji, P., & Pal, S. K.: Rough set based generalized fuzzy C -means algorithm and quantitative indices. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(6), 1529-1540 (2007)
7. Mitra, S., Banka, H., & Pedrycz, W.: Rough–fuzzy collaborative clustering. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 36(4), 795-805 (2006)
8. Pawlak, Z.: Rough sets. *International Journal of Parallel Programming*, 11(5), 341-356 (1982)
9. Yager, R. R.: Some aspects of intuitionistic fuzzy sets. *Fuzzy Optimization and Decision Making*, 8(1), 67-90 (2009)
10. Yang, X. S.: Firefly algorithms for multimodal optimization. In *International symposium on stochastic algorithms* (pp. 169-178). Springer Berlin Heidelberg (2009, October)
11. Zadeh, L. A.: Fuzzy sets. *Information and control*, 8(3), 338-353 (1965)