“BRAIN STORM OPTIMIZATION ALGORITHM”

PROJECT REPORT

Submitted for the course:

DATA MINING (CSE 3019)

By

Aditya Firoda	16BCE2184
Harish Agarwal	16BCE0123
Suyash Pasari	16BCE0624

SLOT: G1(L33+34)

Name of faculty: **Rishin Haldar**

CERTIFICATE

This is to certify that the project work entitled “**BRAINSTORM OPTIMIZATION ALGORITHM**” that is being submitted by “**Aditya Firoda**”, “**Suyash Pasari**” and “**Harish Agarwal**” for the course **Data Mining(CSE3019)** is a record of bonafide work done under my supervision. The contents of this project has been , in full or in parts, not been taken from any other CAL course.

RISHIN HALDAR

Table of Contents

CHAPTERS	NAME
Chapter 1	Abstract
Chapter 2	Introduction
Chapter 3	Brainstorming Process
Chapter 4	Brainstorming Algorithm
Chapter 5	Experiments and Discussion
Chapter 6	Code Implementation
Chapter 7	Conclusion and References

CHAPTER-1

ABSTRACT:

Human being is the most intelligent animal in this world. Intuitively, optimization algorithm inspired by human being creative problem solving process should be superior to the optimization algorithms inspired by collective behavior of insects like ants, bee, etc. In this report, we introduce a novel brain storm optimization algorithm, which was inspired by the human brainstorming process. Two benchmark functions were tested to validate the effectiveness and usefulness of the proposed algorithm.

As we all may have experienced that when we face a difficult problem which every single person can't solve, a group of persons, especially with different background, get together to brain storm, the problem can usually be solved with high probability. Great and un-expectable intelligence can occur from interactive collaboration of human beings. One way to help human beings to interactively collaborate to generate great ideas is to get together a group of people to brainstorm. Optimization technique has become a significant and promising tool in solving various scientific and engineering problems nowadays.

BSO transplants the brainstorming process in human being into optimization algorithm design and gains successes. BSO generally uses the grouping, replacing, and creating operators to produce ideas as many as possible to approach the problem global optimum generation by generation. We will be seeing the process and the steps involved in the algorithm.

CHAPTER-2

INTRODUCTION:

Population-based optimization algorithms have been widely accepted and successfully applied to solve a lot of optimization problems. Unlike traditional single-point based algorithms such as hill-climbing algorithms, a population-based optimization algorithm consists of a set of points (population) which solve the problem through information sharing to cooperate and/or compete among themselves. So far, there are a lot of population-based algorithms existed. The very first population-based algorithms are evolutionary algorithms including evolutionary programming, genetic algorithm, evolution strategy, and genetic programming [EBERHART2007], which were inspired by biological evolution. Recently, there occurred more population-based algorithms which are usually called nature-inspired optimization algorithms instead of evolution-inspired algorithms. Many of nature-inspired optimization algorithms are categorized as swarm intelligence algorithms. In a swarm intelligence algorithm, each individual in the population represents a simple object such as ant, bird, fish, etc. There exist a lot of different swarm intelligence algorithms, among which are particle swarm optimization (PSO) [SHI1998], ant colony optimization algorithm(ACO) [DORIGO1996], bacterial forging optimization algorithm(BFO) [PASSINO2010], firefly optimization algorithm [YANG2008], artificial immune system [CASTRO1999], and etc. In a swarm intelligence algorithm, it is the collective behavior of all individuals that makes the algorithm to be effective in problem optimization. All individuals cooperate and collectively move toward the better and better areas in the solution search space. These individuals represent only simple objects such as birds in PSO, ants in ACO, bacteria in BFO, etc. Human beings are social animals and are the most intelligent animals in the world. Therefore, it is natural to expect that an optimization algorithm inspired by human creative problem solving process will be a good optimization algorithm. In this project, we will introduce a novel optimization algorithm inspired by the human idea generation process – brainstorming process. The remaining paper is organized as follows. In Section 2, the human brainstorming process is reviewed. In Section 3, the novel optimization algorithm inspired by human brainstorming process is introduced and described in detail, followed by experimental simulation and result discussion on two benchmark functions in Section 4. Finally, conclusions are given in Section 5.

CHAPTER-3

As we all may have experienced that when we face a difficult problem which every single person can't solve, a group of persons, especially with different background, get together to brain storm, the problem can usually be solved with high probability. Great and un-expectable intelligence can occur from interactive collaboration of human beings. One way to help human beings to interactively collaborate to generate great ideas is to get together a group of people to brainstorm. A brainstorming process generally follows the steps listed below.

Steps in a Brainstorming Process

Step 1: Get together a brainstorming group of people with as diverse background as possible;

Step 2: Generate many ideas according to the rules in Table 2;

Step 3: Have several, say 3 or 5, clients act as the owners of the problem to pick up several, say one from each owner, ideas as better ideas for solving the problem;

Step 4: Use the ideas picked up in the Step 3 with higher probability than other ideas as clues, and generate more ideas according to the rules in Table 2;

Step 5: Have the owners to pick up several better ideas generated as did in Step 3;

Step 6: Randomly pick an object and use the functions and appearance of the object as clues, generate more ideas according to the rules in Table 2;

Step 7: Have the owners to pick up several better ideas;

Step 8: Hopefully a good enough solution can be obtained by considering and/or merging the ideas generated.

In a brainstorming process, usually there are a facilitator, a brainstorming group of people, and several owners of the problem to be solved. The role of the facilitator is to facilitate the idea generation (brainstorming) process by enforcing the brainstorming group to obey the Osborn's original four rules of idea generation in a brainstorming process [SMITH2002]. The four rules are listed in Table 2 below. The facilitator should not be involved in generating ideas itself, but facilitating the brainstorming process only. The guideline for selecting facilitator is to have a facilitator to have facilitation experience but have less expertise on the background knowledge related to the problem to be solved as possible. The purpose of this is to have generated ideas to have less, if not none, biases from the facilitator.

Osborn's Original Rules for Idea Generation in a Brainstorming Process:

Rule 1. Suspend Judgment

Rule 2. Anything Goes

Rule 3. Cross-fertilize (Piggyback)

Rule 4. Go for Quantity

The Rule 1 says that there is no idea as bad idea. All ideas are good ideas. It is unwise to judge whether a proposed idea is a good or bad idea. Any judgment or criticism must be held back until at least the end of the brainstorming process. The Rule 2 says that anything coming to your mind during the brainstorming process is an idea worth to be shared and recorded. Don't let any idea or thought ignored. The Rule 3 says that lot of ideas can and should be based on ideas already generated. Any generated idea can and should serve as a clue to generate more ideas. The Rule 4 says that it is necessary to generate as many ideas as possible. We first go for quantity of generated ideas. The quality will come from quantity naturally. Without generating large quantity of ideas, it is difficult, if not impossible, to come out ideas with good quality. The purpose to generate ideas according to rules mentioned above is to generate ideas as diverse as possible so that the people in the brainstorming group will be open-minded as much as possible. The operation of picking up an object in Step 6 serves for the same purpose as generating diverse and different ideas. It can help brainstorming group to diverge from previously generated ideas therefore to avoid being trapped by the previously generated ideas. As a consequence, the brainstorming group will be more open-minded and generate more diverse ideas. The problem owners serve for one different purpose. Picking up several good ideas from ideas generated so far is to cause the brainstorming group to pay more attention to the better ideas which the brainstorming group believes to be. The ideas picked-up work like point-attractors for the idea generation process.

CHAPTER-4

The brainstorming process has been successfully applied to generate ideas to solve very difficult and challenging problems. Intuitively, an optimization algorithm designed based on the human being idea generation process should be superior to optimization algorithms inspired by collective behaviors of animals because human beings are the most intelligent animals in the world. The novel optimization algorithm inspired by brainstorming process is given in Table 3. In the procedure of the Brain Storm Optimization (BSO) algorithm shown in the Table 3, the Step 1 is the initialization step as that in other population-based algorithms; the Step 2, 3, and 4 in Table 3 serves the purpose of Step 3, 5, and 7 in Table 1 to pick up several better ideas; the Step 5 in Table 3 simulates the operation of Step 6 in Table 1 to make the population to cover unexplored areas; the Step 6 in Table 3 simulates the idea generation in Step 2, 4, and 6 of Table 1; the Step 6.b simulates generating new idea inspired by one single existing idea while the Step 6.c simulates generating new idea inspired by two existing ideas from two different idea clusters, respectively; the cluster center has more chances to be used to generate new ideas than other ideas in each cluster; certainly a new idea can also be inspired by more than two existing ideas, but it is not simulated in the BSO algorithm for keeping the algorithm simple; the number of clusters in Table 3 serves as the role of problem owners in Table 1; the cluster center in each cluster serves the purpose of better ideas picked up by problem owners. The Step 6.d serves the purpose of keeping better ideas generated. The population size n simulates the number of ideas generated in each round of idea generation in the brainstorming process. For the simplicity of the algorithm, the population size usually is set to be a constant number for all iterations in the BSO algorithm.

Table 3. Procedure of Brain Storm Optimization Algorithm

1. Randomly generate n potential solutions (individuals);

2. Cluster n individuals into m clusters;

3. Evaluate the n individuals;

4. Rank individuals in each cluster and record the best individual as cluster center in each cluster;

5. Randomly generate a value between 0 and 1; a) If the value is smaller than a pre-determined probability p5a, i. Randomly select a cluster center; ii. Randomly generate an individual to replace the selected cluster center;

6. Generate new individuals a) Randomly generate a value between 0 and 1; b) If the value is less than a probability p6b, i. Randomly select a cluster with a probability p6bi; ii. Generate a random value between 0 and 1; iii. If the value is smaller than a pre-determined probability p6biii, 1) Select the cluster center and add random values to it to generate new individual. iv. Otherwise randomly select an individual from this cluster and add random value to the individual to generate new individual. c) Otherwise randomly select two clusters to generate new individual i. Generate a random value; ii. If it is less than a pre-determined probability p6c, the two cluster centers are combined and then added with random values to generate new individual; iii. Otherwise, two individuals from each selected cluster are randomly selected to be combined and added with random values to generate new individual. d) The newly generated individual is compared with the existing individual with the same individual index, the better one is kept and recorded as the new individual;

7. If n new individuals have been generated, go to step 8; otherwise go to step 6;

8. Terminate if pre-determined maximum number of iterations has been reached; otherwise go to step 2.

CHAPTER-5

Experiments and Discussions :

The BSO procedure can be implemented in various ways by setting up BSO algorithm’s parameters differently. In Step 6.b.i of the BSO procedure shown in Table 3, a cluster is selected with probability p6bi, which is proportional to the number of individuals in the cluster. This is, the more individuals a cluster contains, the more likely it will be selected. The Gaussian random values will be used as random values which are added to generate new individuals. The new individual generation in Step 6 can be represented as

$$Xnewd = Xselectedd + \xi * n(\mu, \sigma) \qquad (1)$$

whereXselectedd is the dth dimension of the individual selected to generate new individual; Xnewd is the dth dimension of the individual newly generated; n(μ, σ) is the Gaussian random function with mean μ and variance σ; the ξ is a coefficient that weights the contribution of the Gaussian random value. For the simplicity and for the purpose offine tuning, the ξ can be calculated as

$$\xi = \text{logsig}((0.5 * \text{max\_iternation} - \text{current\_iteration})/k) * \text{rand}() \qquad (2)$$

where logsig() is a logarithmic sigmoid transfer function, max_iteration is the maximum number of iterations, and current_iteration is the current iteration number, k is for changing logsig() function's slope, and rand() is a random value within (0,1). For the purpose of validating the effectiveness and usefulness of the proposed BSO algorithm, in this paper, a set of parameters are set intuitively for the procedure of BSO given in Table 3 for both testing functions below. The set of parameters are listed in Table 4 below.

Table 4. Set of Parameters for BSO Algorithm

n m p5a p6b p6biii p6c k Max_ iteration μ σ 100 5 0.2 0.8 0.4 0.5 20 2000 0 1

The BSO algorithm is then tested on two benchmark functions listed in Table 5. We use k-mean cluster algorithm to cluster n individuals into m clusters. For each benchmark function, the BSO will be run 50 times to obtain reasonable statistical results. The first benchmark function is the Sphere function, which is an unimodal function. The simulation results of testing BSO on the Sphere function with dimensions 10, 20, and 30, respectively, are given in Table 6.

Table 5. Benchmark Functions Tested in This Paper

Index Function name Function expression 1 Sphere ∑= = D i i xxf 1 2 1() 2 Rastrigin ∑= −+ = D i ii xxf 1 2 2 10) 2)10cos((π

Table 6. Simulation Results of BSO Algorithm

function dimension mean best worst variance 10 3.82E-44 1.50775E-44 7.12557E-44 1.57592E-88 20 3.1E-43 1.61402E-43 4.56276E-43 4.0471E-87 Sphere 30 1.15E-42 8.07001E-43 1.69603E-42 4.69513E-86 10 3.820643 1.989918 6.964713 1.954026 20 18.06844 8.954632 26.86387 19.65172 Rastrigin 30 32.91322 17.90926 58.70249 82.82522

The results given in the Table 6 are mean, best, worst minimum values and their variance of the final iteration over 50 runs. From the Table 6, it can be observed that very good results can be obtained by the implemented BSO algorithm. The second is the Rastrigin function, which is a multimodal function. The simulation results of testing BSO on the Rastrigin function with dimensions 10, 20, and 30, respectively, are given in Table 6. From Table 6, we can observe that good results can also be obtained by this very version of BSO algorithm.

CHAPTER-6

CODE:

```
#include <bits/stdc++.h>

#include<string>

#include<cstdlib>

using namespace std;

#include<vector>

#define pb push_back

#define eb emplace_back

#define mp make_pair

#define f first

#define s second

#define fast_cin ios_base::sync_with_stdio(false);cin.tie(NULL)

typedef long long ll;

typedef long double ld;

#define fast_cin ios_base::sync_with_stdio(false);cin.tie(NULL)
```

```
// here fitness algorithm can be considered in the form of euclidean distance

int generatecluster(ll N,ll k);

typedef pair<ld,ld> p;

vector<p> vect1,vect2,cls[100],c,clsidea;

ld matter[1000][1000],R[100][100],D[100],DB=0,scatter[1000];

// vect1 store N ideas

// vect2 store M values

// cls M cluster  with grouping

// c each cluster centre

// dist[100] store each of N ideas belong to which cluster

ll dist[100];

ld DBindex(ll N,ll k);

ld dunnindex(ll N,ll k);

fstream file;

string word;

int main()

{

   ll N,k,i,j;

   ld
l,t,n,count1=0,count2=0,index=0,sum=0,temp=0,temp1=0,temp2=0,prob=0,newideax,newideay,fitnessi
dea=0, fitness=0;

   double  p_replace,p_one_centre=0.4,p_two_centre=0.5;

    ll x=0,y=0;

   double x1;
```

```
cin>>N>>k;
 //N=stoi(word);
 //k=atoi(word);


 // The object has the value 12345 and stream
 // it to the integer x
// int x = 0;
 //geek >> k;
 generatecluster(N, k);


 p_replace =0.2;
 x1=rand()%100;
 x1=x1/100.0000;
 // step 6
 if(x1<p_replace)
 {
     temp=rand()%3;
     x=rand()%10;
     y=rand()%10;
     cout<<"random idea in step 6 "<<x<<" "<<y<<endl;
     c[temp]=mp(x,y);


 }
 // step 8 over


 // step 9
```

```
for(i=0;i<N;i++)
{

   ll p_one=0.8;
   x1=rand()%100;
   x1=x1/100.000;

   if(x1<p_one)
   {
      j=rand()%3;

// step 11
      p_one_centre=0.4;
     // prob= rand()%N;
      x1=((rand()%100)/100.000);
     //prob11=prob11/N;
     // step 12

     if(x1<p_one_centre)
     {
        // step 13
        x= vect1[j].f+(rand()%10);
       y=vect1[j].s+(rand()%10);
       //vect1[j]=mp(x,y);
       newideax=x;
       newideay=y;
```

```
         }
         else
         {
             x= vect1[j].f+(rand()%10);
            y=vect1[j].s+(rand()%10);
           // vect1[j]=mp(x,y);
            newideax=x;
            newideay=y;








         }
         // step 15 over


       }
         else
         {
// step 17
// generate new idea based on two cluster
             x=rand()%10;
             y=rand()%10;
             //cout<<"x "<<x<<" y "<<y<<endl;
             temp1=rand()%k;
```

```
    temp2=rand()%k;
    // probability of using cluster center
    p_two_centre=0.5;


     x1=rand()%100;
    x1=x1/100.000;
// step 20
    if(x1<p_two_centre)
        // combine the two selected cluster centers and add
        //with random values to generate new idea Yi
    {


              newideax=((c[temp1].f+c[temp1].f)/2)+x;
              newideay=((c[temp1].s+c[temp1].s)/2)+y;
    }
    // step 21
    else
    {
            newideax=((c[temp1].f+c[temp1].f)/2)+x;
              newideay=((c[temp1].s+c[temp1].s)/2)+y;




    }


    // step 23
```

```
        }
        //clsidea



        // step 25


      // evaluate the idea Yi and replace1 Xi if Yi has better fitness
      //than Xi


       fitnessidea=0;
       fitness=0;
       for(j=0;j<k;j++)
    {


      fitnessidea+=(abs(vect1[i].f-vect2[j].f)+abs(vect1[i].s-vect2[j].s));
      fitness+=(abs(vect1[i].f-newideax)+abs(vect1[i].s-newideay));


    }





2,,,    if(fitnessidea<fitness)
      vect1[i]=mp(newideax,newideay);
```

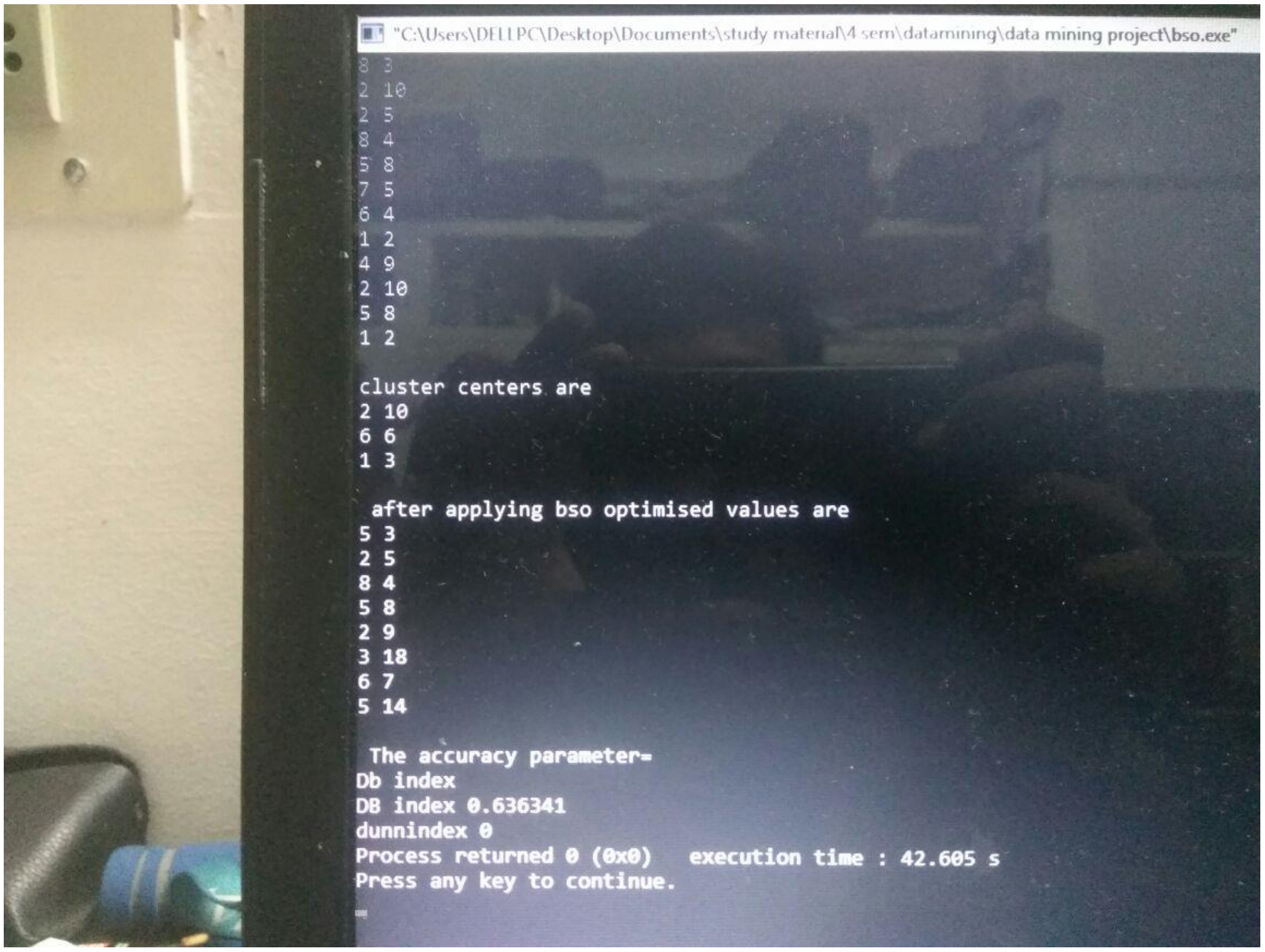
```
 }
    cout<<"\n after applying bso optimised values are "<<endl;
    for(i=0;i<N;i++)
      cout<<vect1[i].f<<" "<<vect1[i].s<<endl;
DBindex(N,k);
dunnindex(N,k);
    return 0;
}
ld dunnindex(ll N,ll k)
{
    ll i,j;
    ld min1=999999.000000,max1=0.0000000,dunnindex=0.00;
    for(i=0;i<k;i++)
        for(j=0;j<k;j++)
        min1=min(matter[i][j],min1);
    //cout<<"\nmin1 "<<min1<<endl;
    for(i=0;i<k;i++)
        max1=max(D[i],max1);
     //cout<<"max1 "<<max1<<endl;
    dunnindex=min1/max1;

    cout<<"\ndunnindex "<<dunnindex;


    return 0;

  return 0;

}

ld DBindex(ll N,ll k)

{

    ll i,j;

    cout<<"\n The accuracy parameter= \n";

cout<<"Db index ";

ld sum=0;

for(i=0;i<k;i++)

    {

        sum=0;

    for(j=0;j<cls[i].size();j++)

         sum+=pow((cls[i][j].s-c[i].s),2)+pow((cls[i][j].f-c[i].f),2);

    sum=sum/cls[i].size();

    sum=pow(sum,.5);

    scatter[i]=(sum);


   // cout<<"\nscatter= "<<scatter[i]<<endl;

    }




for(i=0;i<k;i++)
```

```
      for(j=0;j<k;j++)
           matter[i][j]=pow((pow((c[i].s-c[j].s),2)+pow((c[i].f-c[j].f),2)),.5);
DB=0;
for(i=0;i<k;i++)
    {
     D[i]=0;
      for(j=0;j<k;j++)
        {
            if(matter[i][j]!=0)
         R[i][j]=(scatter[i]+scatter[j])/matter[i][j];
         else
              R[i][j]=0;
          D[i]=max(D[i],R[i][j]);


        }
     //  cout<<"\nD[i]= "<<D[i]<<endl;;
    DB+=D[i];
    }
DB=DB/k;
cout<<"\nDB index "<<DB;
```

```
    return 0;
}

int generatecluster(ll N,ll k)
{
    ll i,j,x,y,n,sum;

    for(i=0;i<N;i++)
            {
                cin>>x>>y;
            /*    file>>word;
    //N=stoi(word);
    x=atoi(word);

      //cin>>N>>k;
    file>>word;
    //N=stoi(word);
    y=atoi(word);*/
               // cin>>x>>y;
                vect1.pb(mp(x,y));

            }
for(i=0;i<k;i++)
            {
                cin>>x>>y;
                /*file>>word;
    //N=stoi(word);

    x=atoi(word);


      //cin>>N>>k;

    file>>word;

    //N=stoi(word);

    y=atoi(word);*/


    // The object has the value 12345 and stream

    // it to the integer x


   // geek >> y;

                vect2.pb(mp(x,y));

            }


for(i=0;i<N;i++)

{

    x=99999;


         for(j=0;j<k;j++)

    {

        sum=0;

        sum+=(abs(vect1[i].f-vect2[j].f)+abs(vect1[i].s-vect2[j].s));

        if(x>sum)

        {

            x=sum;
            dist[i]=j+1;


        }
    }


}


for(i=1;i<=N;i++)
    cls[dist[i-1]-1].pb(mp(vect1[i-1].f,vect1[i-1].s));
  // cout<<dist[i-1]<<" "<<vect1[i-1].f<<" "<<vect1[i-1].s<<endl;
x=0;
y=0;


for(i=0;i<k;i++)
  {
      x=0;
      y=0;


      n=cls[i].size();
      for(j=0;j<n;j++)
        {
          x+=cls[i][j].f;
          y+=cls[i][j].s;
        }
      c.pb(mp(x/n,y/n));
```

```
}
cout<<"\ncluster centers are \n";
   for(i=0;i<k;i++)
      cout<<c[i].f<<" "<<c[i].s<<endl;









   return 0;
}
```

OUTPUT:



```
"C:\Users\DELLPC\Desktop\Documents\study material\4 sem\datamining\data mining project\bso.exe"
8 3
2 10
2 5
8 4
5 8
7 5
6 4
1 2
4 9
2 10
5 8
1 2

cluster centers are
2 10
6 6
1 3

 after applying bso optimised values are
5 3
2 5
8 4
5 8
2 9
3 18
6 7
5 14

 The accuracy parameter=
Db index
DB index 0.636341
dunnindex 0
Process returned 0 (0x0)   execution time : 42.605 s
Press any key to continue.
```

CHAPTER-7

CONCLUSION:

In this project report, we introduced a novel brain storm optimization algorithm which was inspired by the human brainstorming process. Human beings are the most intelligent animals in the world, therefore, it is natural to believe that the optimization algorithm inspired by collective behavior of human beings should be superior to the optimization algorithms inspired by collective behavior of injects such as ants, birds, etc. The proposed BSO algorithm was implemented and tested on two benchmark functions. Even though the simulation results are very preliminary, the results DID validate the effectiveness and usefulness of the proposed BSO algorithm which is the purpose of this paper. More detailed analysis and experimental tests are our next step research work, e.g. using different clustering algorithm, using different random functions for generating new individuals, etc.

REFERENCES:

1. de Castro, L.N., Von Zuben, F.J.: Artificial Immune Systems: Part I -Basic Theory and Applications, School of Computing and Electrical Engineering, State University of Campinas, Brazil, No. DCA-RT 01/99 (1999)

2. Dorigo, M., Maniezzo, V., Colorni, A.: The ant system: Optimization by a colony of cooperating agents. IEEE Trans. Syst., Man, Cybern. B 26(2), 29–41 (1996)

3. Eberhart, R.C., Shi, Y.: Computational Intelligence, Concepts to Implementation, 1st edn. Morgan Kaufmann Publishers, San Francisco (2007)

4. Passino, K.M.: Bacterial Foraging Optimization. International Journal of Swarm Intelligence Research 1(1), 1–16 (2010)

5. Shi, Y., Eberhart, R.C.: A Modified Particle Swarm Optimizer. In: 1998 IEEE International Conference on Evolutionary Computation, Anchorage, Alaska, USA, May 4-9 (1998)

6. Smith, R.: The 7 Levels of Change, 2nd edn. Tapeslry Press (2002)

7. Yang, X.: Nature-Inspired Metaheuristic Algorithms. Luniver Press (2008)

8. W. N. Chen, J. Zhang, H. Chung, W. L. Zhong, W. G. Wu, and Y. H. Shi, “A novel set-based particle swarm optimization method for discrete optimization problems,” IEEE Trans. Evol. Comput., vol. 14, no. 2, pp. 278-300, April 2010.

9. Y. Shi, “Brain storm optimization algorithm,” in Proc. 2nd Int. Conf. on Swarm Intelligence, 2011, pp. 303-309.

10. F. Peng, K. Tang, G. L. Chen, and X. Yao, “Population-based algorithm portfolios for numerical optimization,” IEEE Trans. Evol. Comput., vol. 14, no. 5, pp. 782-800, Oct. 2010.