

# Evolutionary Algorithms in Image Processing

Presented By

**R.V.V. Krishna**

Associate Professor & HOD

ECE Department -1

Aditya College of Engineering & Technology

# Outline

- ❑ Introduction
- ❑ When Evolutionary Algorithms are useful.
- ❑ Popular Evolutionary Algorithms
- ❑ Genetic Algorithm
- ❑ Differential Evolution
- ❑ Teaching Learning Based Optimization
- ❑ Conclusion

# Introduction

An evolutionary algorithm (EA) is a generic population-based metaheuristic optimization algorithm.

An EA uses mechanisms inspired by biological evolution, such as crossover, mutation, recombination, and selection.

Candidate solutions to the optimization problem play the role of individuals in a population, and the fitness function determines the quality of the solutions .

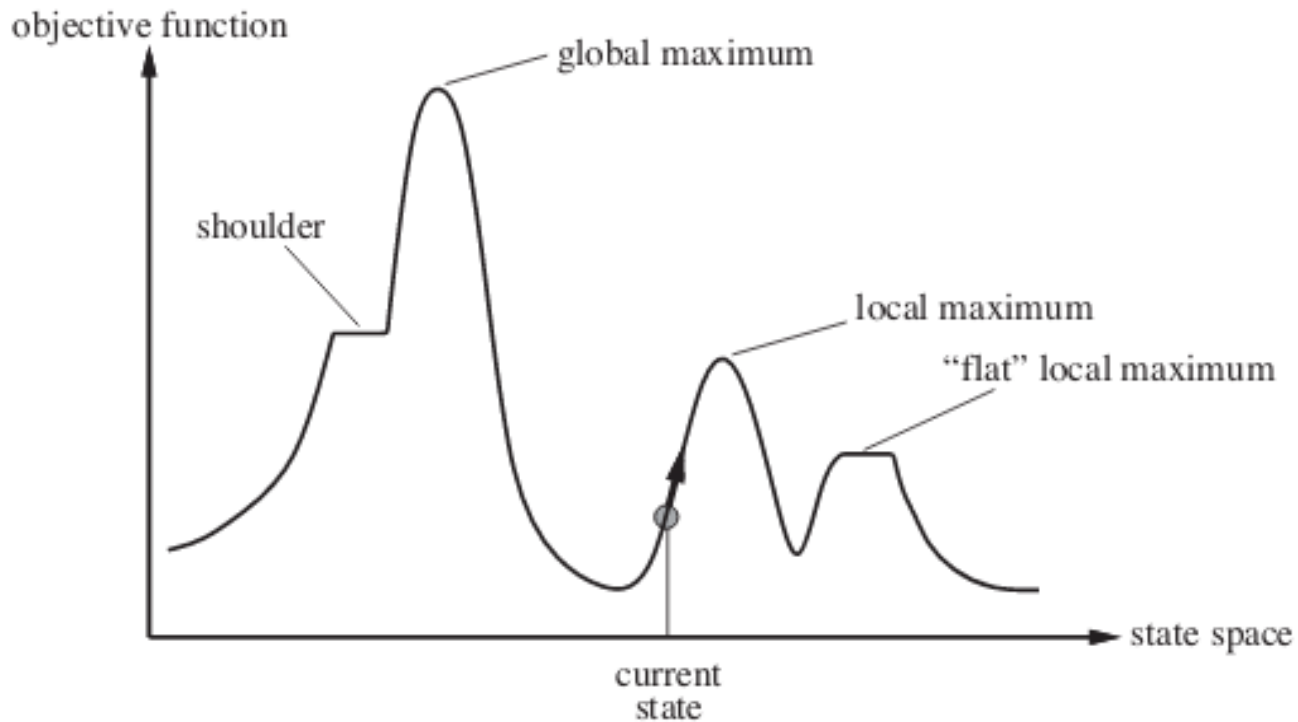
## When Evolutionary algorithms are useful?

Evolutionary algorithms are typically used to provide good approximate solutions to problems that cannot be solved easily using other techniques.

It may be too computationally-intensive to find an exact solution but sometimes a near-optimal solution is sufficient. In these situations evolutionary techniques can be effective.

Due to their random nature, evolutionary algorithms are never guaranteed to find an optimal solution for any problem, but they will often find a good solution if one exists.

Simple optimization algorithms such as Gradient descent algorithm and Hill climbing algorithms get stuck in local minima or local maxima



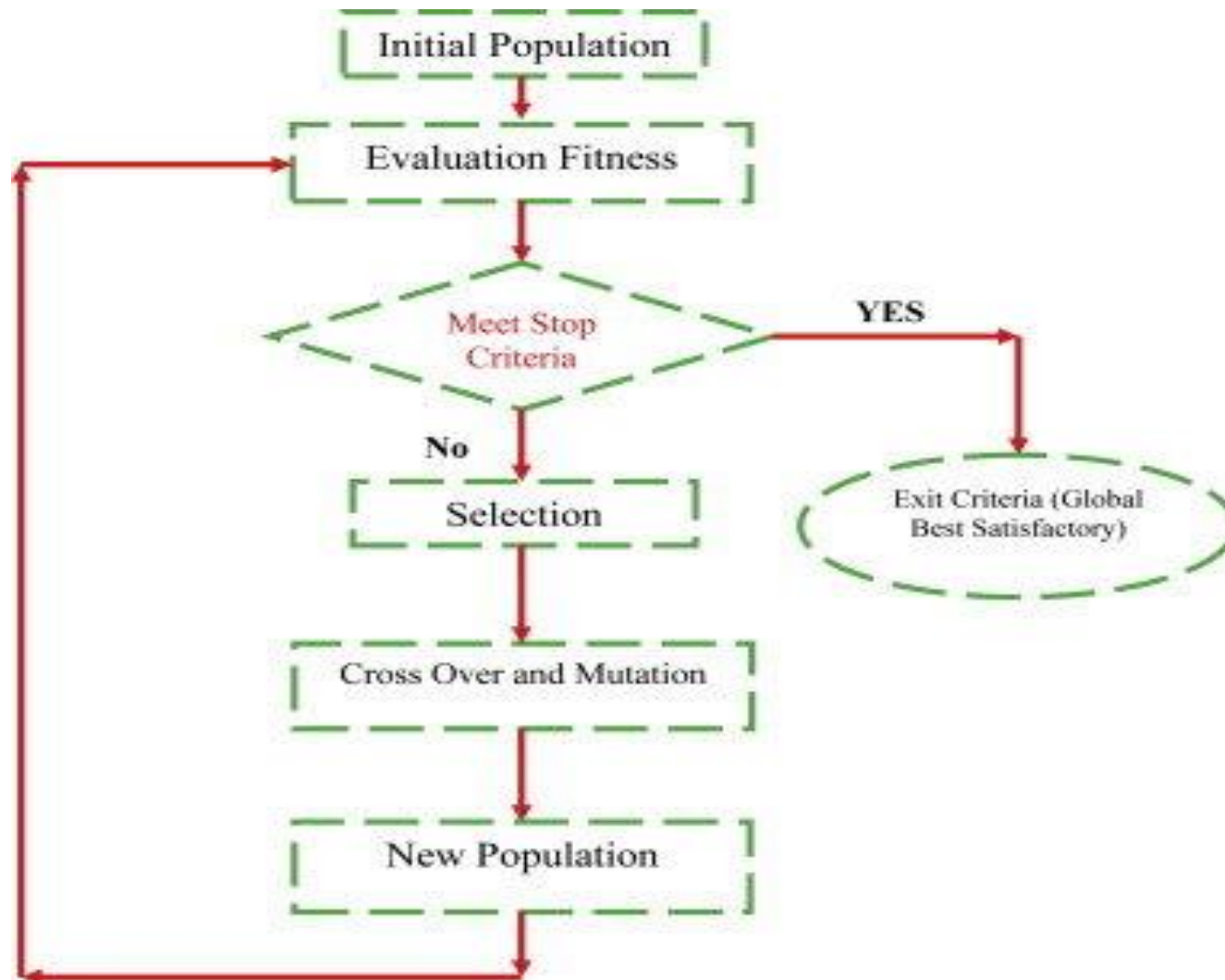
# Some Popular Evolutionary Algorithms

1. Genetic Algorithm (D.E.Goldberg, 1989)
2. Differential Evolution (DE) (Storn & Price, 1995)
3. Teaching Learning Based Optimization (R.V.Rao et al., 2011)
4. Simulated annealing (Kirkpatrick et al. 1983)
5. Ant colony optimization (Dorigo, 1992)
6. Particle swarm optimization (Kennedy & Eberhart 1995)
7. Harmony search (Geem, Kim & Loganathan 2001)
8. Artificial bee colony algorithm (Karaboga 2005)
9. Cat Swarm Optimization (Chu, Tsai, and Pan 2006)
10. River formation dynamics (Rabanal, Rodríguez & Rubio 2007)
11. Gravitational search algorithm (Rashedi, Nezamabadi-pour & Saryazdi 2009)
12. Cuckoo search (Yang & Deb 2009)
13. Artificial swarm intelligence (Rosenberg 2014)
14. Killer Whale Algorithm (Biyanto 2016)
15. Rain Water Algorithm (Biyanto 2017)
16. Mass and Energy Balances Algorithm (Biyanto 2018)
17. Hydrological Cycle Algorithm (Wedyan et al. 2017)
18. Emperor Penguins Colony (Harifi et al. 2019)
19. Momentum Balance Algorithm (MBA) (Biyanto et al. 2019)

# Genetic Algorithm:

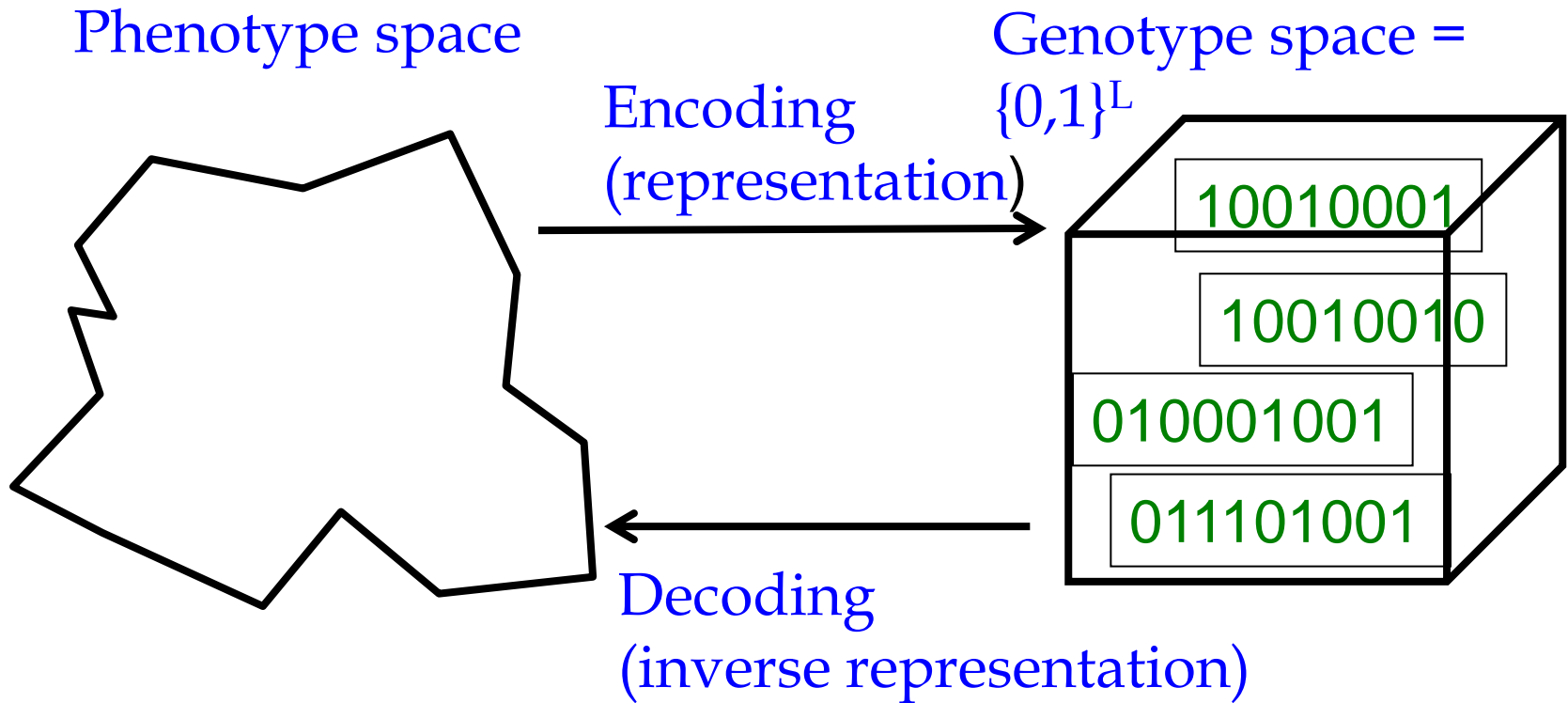
- Developed: USA in the 1970's
- Early names: J. Holland, K. DeJong, D. Goldberg
- Typically applied to:
  - discrete optimization
- Attributed features:
  - not too fast
  - good heuristic for combinatorial problems
- Special Features:
  - Traditionally emphasizes combining information from good parents (crossover)
  - many variants, e.g., reproduction models, operators

# Flow Chart of Genetic Algorithm





|                    |   |
|--------------------|---|
| Representation     | Binary strings                              |
| Recombination      | N-point or uniform                          |
| Mutation           | Bitwise bit-flipping with fixed probability |
| Parent selection   | Fitness-Proportionate                       |
| Survivor selection | All children replace parents                |
| Speciality         | Emphasis on crossover                       |

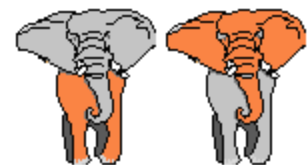
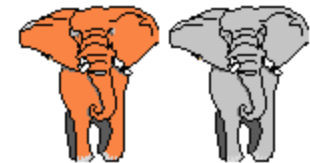
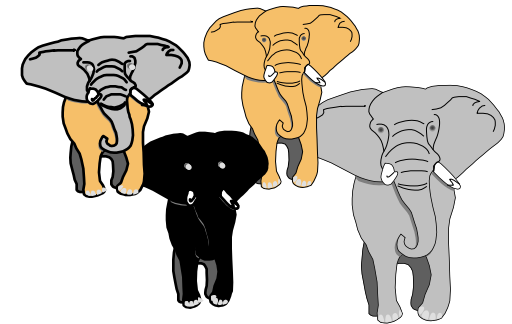
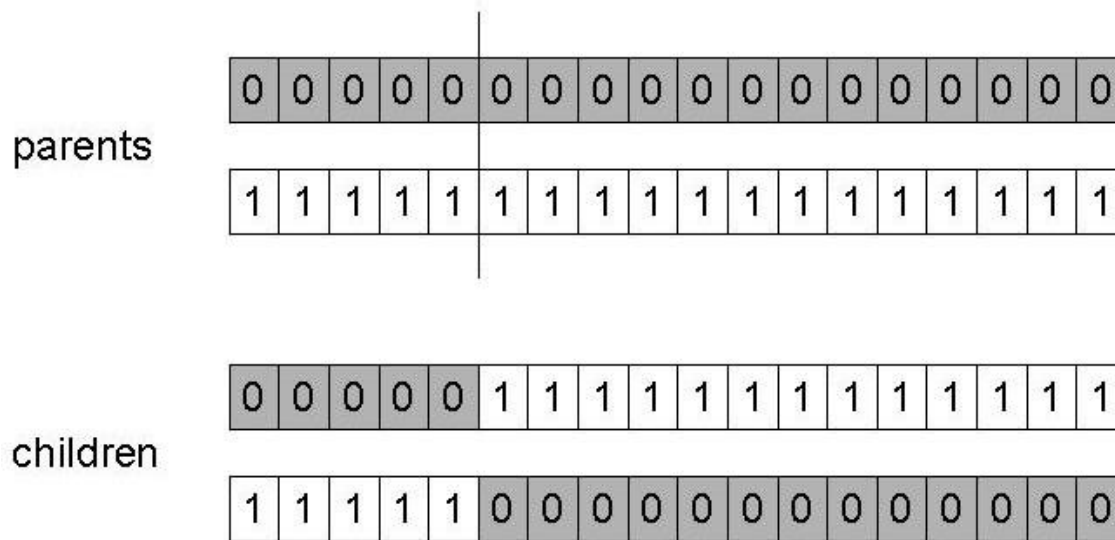


# GA Reproduction Cycle

1. Select parents for the mating pool  
(size of mating pool = population size)
2. Shuffle the mating pool
3. For each consecutive pair apply crossover with probability  $p_c$  , otherwise copy parents
4. For each offspring apply mutation (bit-flip with probability  $p_m$  independently for each bit)
5. Replace the whole population with the resulting offspring

# Single Point Cross Over

- Choose a random point on the two parents
- Split parents at this crossover point
- Create children by exchanging tails
- $P_c$  typically in range (0.6, 0.9)



# Mutation

- Alter each gene independently with a probability  $p_m$
- $p_m$  is called the mutation rate
  - Typically between  $1/\text{pop\_size}$  and  $1/\text{chromosome\_length}$

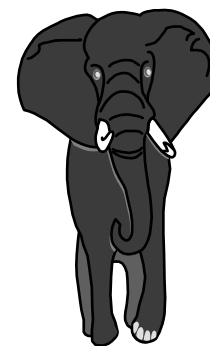
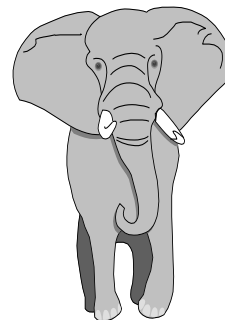
before

1 1 1 1 1 1 1

after

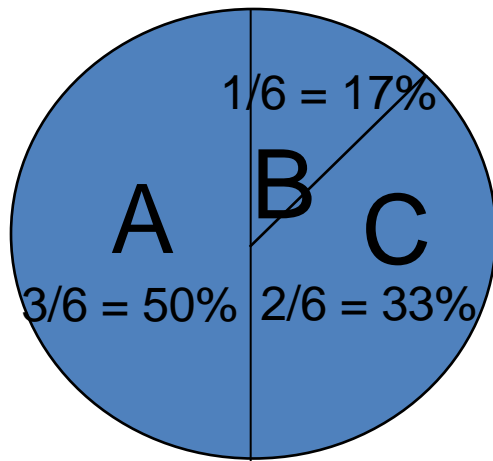
1 1 1 0 1 1 1

mutated gene



# Selection

- Main idea: better individuals get higher chance
  - Chances proportional to fitness
  - Implementation: roulette wheel technique
    - » Assign to each individual a part of the roulette wheel
    - » Spin the wheel n times to select n individuals



$\text{fitness}(\text{A}) = 3$

$\text{fitness}(\text{B}) = 1$

$\text{fitness}(\text{C}) = 2$



Boltzman selection, tournament selection, rank selection, steady state selection and some others.

## Crossover OR mutation?

- Decade long debate: which one is better / necessary / main-background
- Answer (at least, rather wide agreement):
  - it depends on the problem, but
  - in general, it is good to have both
  - both have another role
  - mutation-only-EA is possible, crossover-only-EA would not work

**Exploration:** Discovering promising areas in the search space, i.e. gaining information on the problem

**Exploitation:** Optimising within a promising area, i.e. using information

There is co-operation AND competition between them

- Crossover is explorative, it makes a *big* jump to an area somewhere “in between” two (parent) areas
- Mutation is exploitative, it creates random *small* diversions, thereby staying near (in the area of ) the parent



## A Simple Example

Let us consider the optimization problem for maximizing the function

$f(x) = x + 10 \sin(5x) + 7 \cos(4x) + \sin(x)$ , subject to the constraint  $x$  varies from 0 to 9.

The numbers between 0 and 9 with the resolution of 0.001 are treated as the chromosomes used in the Genetic Algorithm. (i.e) Float chromosomes are used in this example.

Population size of 10 chromosomes is survived in every generation. Arithmetic cross over is used as the genetic operator.

Mutation operation is not used in this example

Roulette Wheel selection is made at every generation. Algorithm flow is terminated after attaining the maximum number of iterations. In this example Maximum number of iterations used is 100.

```

>> clear all, close all
>> pop=0:0.001:9;
>> pos=round(rand(1,10)*9000)+1;
>> pop1=pop(pos);
>> BEST=[];
>> for iter=1:1:100
>> col1=[];
>> col2=[];
>> for do=1:1:10
>> r1=round(rand(1)*9)+1;
>> r2=round(rand(1)*9)+1 ;
>> r3=rand;
>> v1=r3*pop1(r1)+(1-r3)*pop1(r2);
>> v2=r3*pop1(r2)+(1-r3)*pop1(r1);
>> col1=[col1 v1 v2];
>> end
>> sect=fcn(col1)+abs(min(fcn(col1)));
>> sect=sect/sum(sect);
>> [u,v]=min(sect);
>> c=cumsum(sect);

```

% Initial population (10 solutions)

% 100 iterations

% crossover and generate  
% 20 solutions

% fitness evaluation  
% fitness normalization

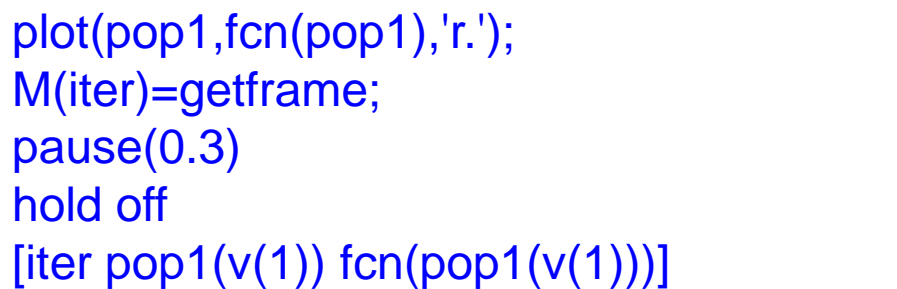
% roulette wheel

```

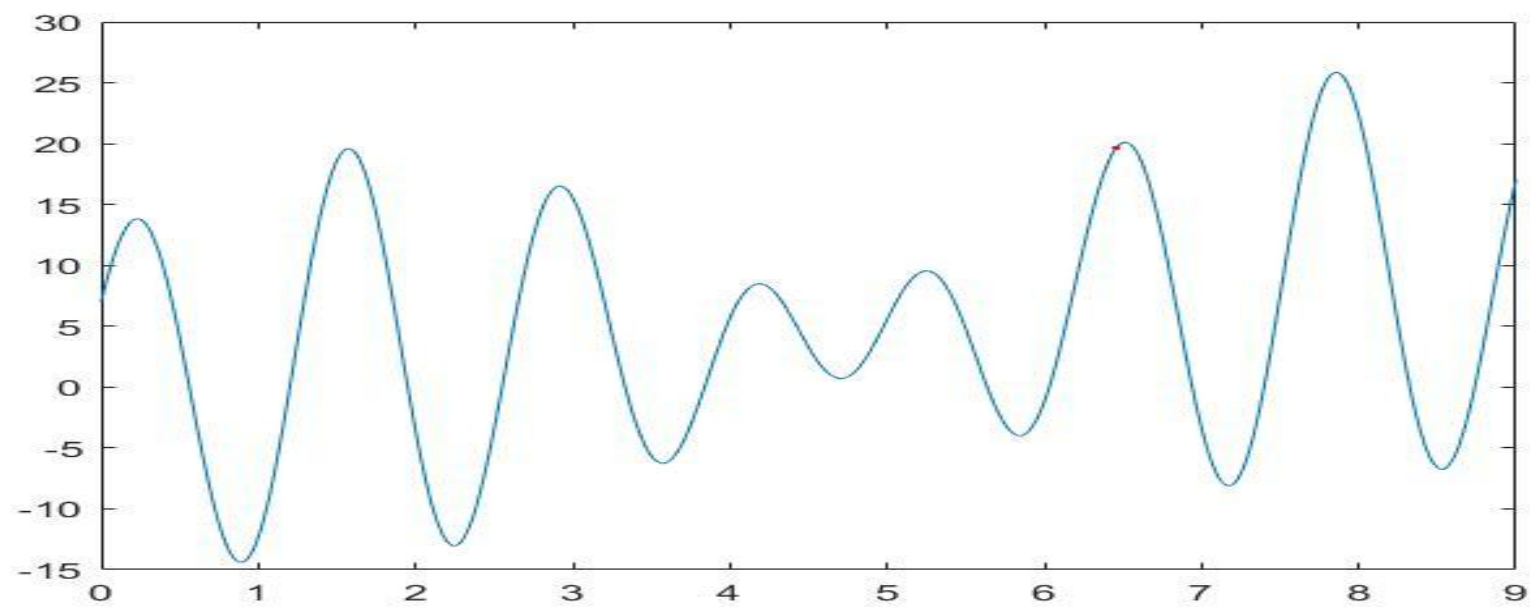
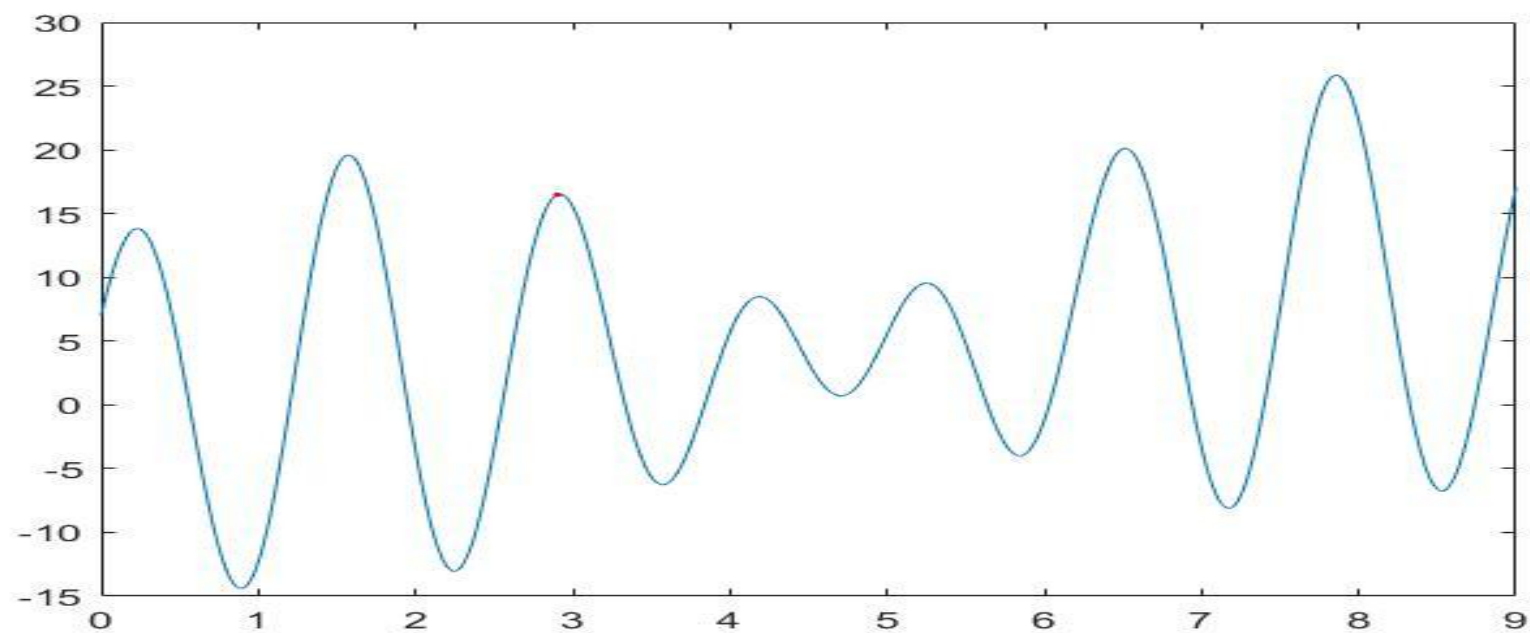
>> for i=1:1:10
>> r=rand;
>> c1=c-r;
>> [p,q]=find(c1>=0);           % Selection of chromosomes for next iteration
>> if(length(q)~=0)
>> col2=[col2 col1(q(1))];
>> else
>> col2=[col2 col1(v)];
>> end
>> end
>> pop1=col2;                   % update the parent for next generation
>> s=fcn(pop);
>> plot(pop,s)
>> [u,v]=max(fcn(pop1));
>> BEST=[BEST;pop1(v(1)) fcn(pop1(v(1)))]; % Record the best solution
>> hold on
>> plot(pop1,fcn(pop1),'r. ');
>> M(iter)=getframe;
>> pause(0.3)
>> hold off
>> [iter pop1(v(1)) fcn(pop1(v(1)))]
>> end

```

```
pop1=col2;  
s=fcu(pop);  
plot(pop,s)  
[u,v]=max(fcu(pop1));  
BEST=[BEST;pop1(v(1)) fcu(pop1(v(1)))];  
hold on  
plot(pop1,fcu(pop1),'r.');
```



```
M(iter)=getframe;  
pause(0.3)  
hold off  
[iter pop1(v(1)) fcu(pop1(v(1)))]  
end  
for i=1:1:14  
D(:, :, i)=M(i).cdata;  
end  
figure  
imshow(M(1).cdata)  
figure  
imshow(M(4).cdata)  
figure  
imshow(M(10).cdata)  
figure  
imshow(M(30).cdata)
```

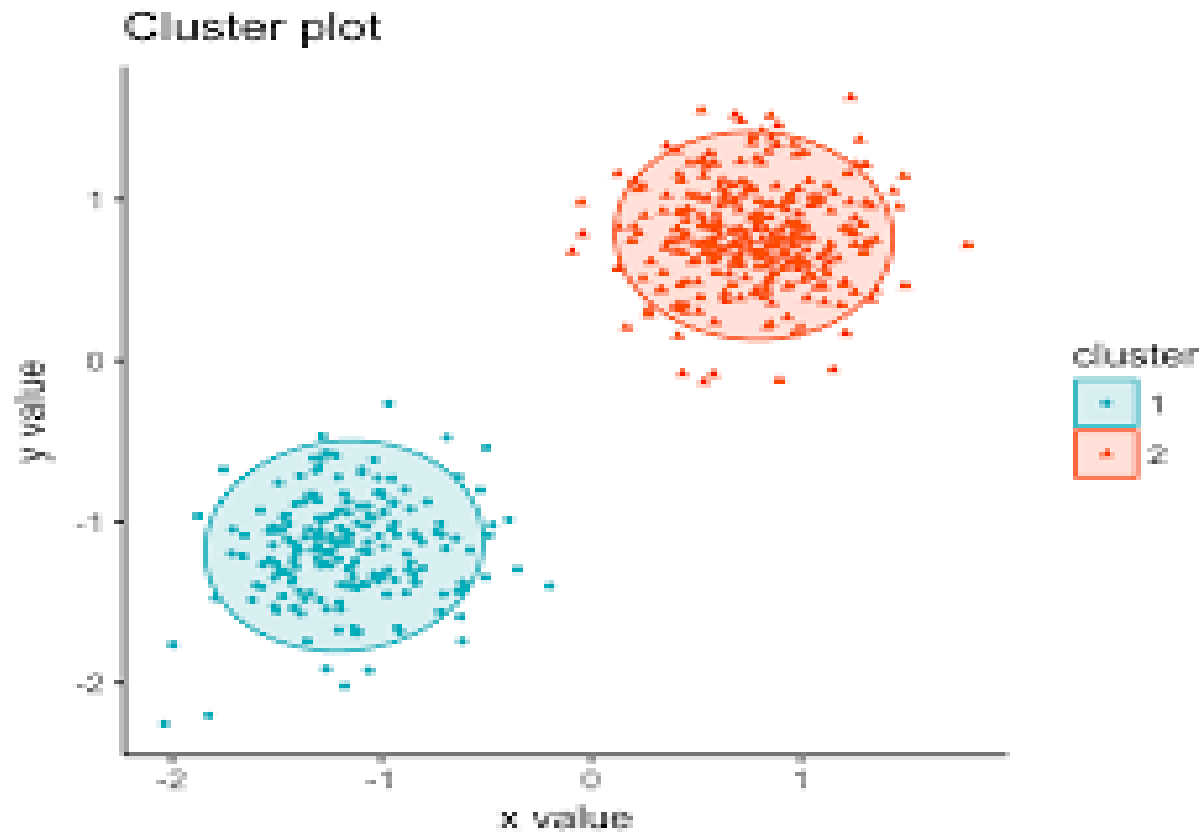


Note that Genetic algorithm ends up with local maxima as shown in the figure. This is the drawback of the Genetic Algorithm

When you run the algorithm again, it may end up with Global maxima. The chromosomes generated randomly during the first generation affects the best solution obtained using genetic algorithm.

Best chromosome at every generation is collected. Best among the collection is treated as the final Best solution which maximizes the function  $f(x)$ .

# Application of Evolutionary Algorithms in data clustering



# How can we tell if our clustering is any good?

- Davies-Bouldin Index

*A function of the ratio of the sum of within-cluster scatter to between-cluster separation*

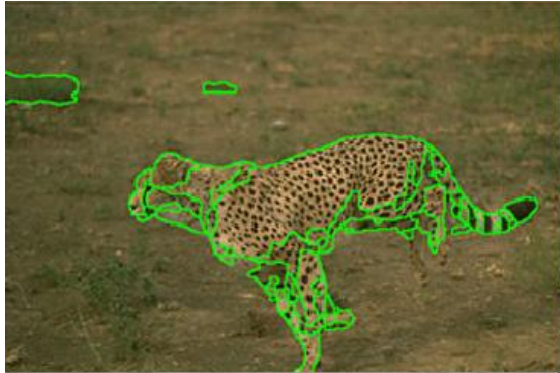
$$DB = \frac{1}{n} \sum_{i=1}^n \max_{i \neq j} \left\{ \frac{S_n(Q_i) + S_n(Q_j)}{S(Q_i, Q_j)} \right\}$$

where:  $n$  is the number of clusters,  $S_n$  is the average distance of all objects from the cluster to their respective cluster centre, and  $S(Q_i, Q_j)$  is the distance between cluster centres

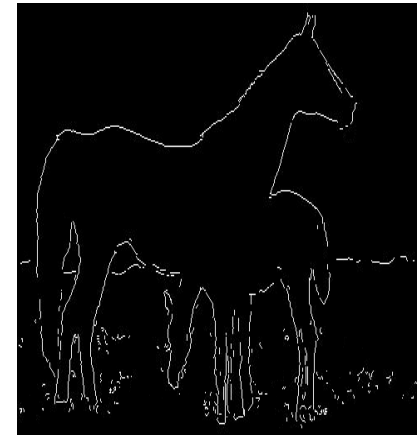
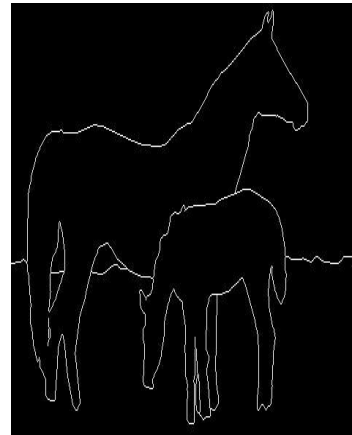
Small values are therefore indicative of good clustering



# Segmentation of color Images



Color Texture Images



Original Image

Ground Truth

Segmented Image

# Teaching Learning Based Optimization :

- Teaching-learning-based optimization (TLBO), is proposed by R.V. Rao et al 2011 ; It is inspired from the philosophy of teaching and learning.
- TLBO is based on the effect of the influence of a teacher on the output of learners in a class which is considered in terms of results or grades.
- The process of working of TLBO is divided into two parts. The first part consists of “Teacher Phase” and the second part consists of “Learner Phase”.
- The “Teacher Phase” means learning from the teacher and the “Learner Phase” means learning through the interaction between learners.

## Teacher Phase :

Let Teacher and Mean be the existing best solution and the mean solution of all learners of the class at any iteration, respectively. The difference between Teacher and Mean is given by

$$\text{Difference\_Mean} = r \times (\text{Teacher} - \text{TF} \times \text{Mean})$$

$$\text{New } X_i = X_i + \text{Difference\_Mean}$$

TF is the teaching factor which decides the value of mean to be changed, and  $r$  is the random number in the range  $[0, 1]$ . Value of TF can be either 1 or 2. The value of TF is decided randomly with equal probability as,

$$\text{TF} = \text{round}[1 + \text{rand}(0,1)\{2,-1\}]$$

The new  $X_i$  is accepted if it gives a better function value

## Learner Phase :

It is the second part of the algorithm where learners increase their knowledge by interaction among themselves.

A learner interacts randomly with other learners with the help of group discussions, presentations, formal communications, etc. A learner  $X_i$  learns something new if the other  $X_j$  learner has more knowledge than him or her

$$\begin{aligned} \text{New } X_i &= X_i + r \times (X_i - X_j) && \text{if } f(X_i) > f(X_j) \\ &= X_i + r \times (X_j - X_i) && \text{otherwise} \end{aligned}$$

The new  $X_i$  is accepted if it gives a better function value

```

Initialize  $P = (x_1, x_2, \dots, x_N)$ ; (N points in D)
While ( fitness value  $\neq$  termination criteria )
{
for  $i := 1$  to N do
for  $j := 1$  to N do
compute new population;

```

Teacher Phase:

```

New  $X_i = X_i + \text{Difference\_Mean}$ 
Difference_Mean =  $r \times (\text{Teacher} - \text{TF} \times \text{Mean})$ 
TF = round  $[1 + \text{rand}(0, 1)]$ 

```

Learner Phase:

```

New  $X_i = X_i + r \times (X_i - X_j)$       if  $f(X_i) > f(X_j)$ 
       $X_i + r \times (X_j - X_i)$       otherwise
end for
}

```

This algorithm requires only the common control parameters such as the population size and the number of generations and does not require any algorithm-specific control parameters.

# Differential Evolution

```
function Differential evolution ( )  
  Initialize  P = (x1, x2, . . . , xN); (N points in D)  
  While ( fitness vale != termination criteria )  
  {  
    for i := 1 to N do  
      compute a mutant vector u;  
      create y by the crossover of u and xi ;  
      if f(y) < f(xi) then insert y into Q  
      else insert xi into Q  
    endif;  
  endfor;  
  P := Q;  
}
```

## Mutation in DE

The  $k^{\text{th}}$  chromosome at generation  $G$ ,  $X_{kG}$  (referred to as the target chromosome in DE nomenclature), DE produces a mutant chromosome

$$V_{kG} = [v_{1kG}, v_{2kG}, \dots, v_{DkG}],$$

where  $D$  is the number of decision variables, through mutation process.

$$DE / rand / 1: V_{k,G} = X_{r_1^k,G} + F(X_{r_2^k,G} - X_{r_3^k,G})$$

where  $r_1^k$ ,  $r_2^k$  and  $r_3^k$  are mutually exclusive and randomly chosen indices from  $[1, NP]$  and are also different from the base index  $k$  (where  $NP$  is the population size).

Here  $F$  is the scaling parameter for amplifying the difference of two chromosomes and is chosen from the interval  $[0, 2]$ .

When  $F$  is small it leads to chromosomes with little deviation, and when  $F$  is large new solutions are explored in the search space.

## Crossover in DE

Crossover operation is used to further supplement the latent diversity of the population. In crossover, the mutant chromosome  $V_{kG}$  interchanges its components with the target chromosome  $X_{kG}$  with a probability  $C_r \in [0,1]$  to form the trial chromosome  $U_{k,G}$

$$U_{k,G} = [U_{1kG}, U_{2kG}, \dots, U_{DkG}]$$

$$u_{j,k,G} = \begin{cases} v_{j,k,G} \Rightarrow (rand_{k,j} \in [0,1] \leq CR \text{ or } j = j_{rand}) \\ x_{j,k,G} \Rightarrow \text{otherwise} \end{cases}$$



# Conclusion

Evolutionary algorithms are explored in many engineering disciplines. Different variants of these algorithms appear in the literature. Hybrid evolutionary algorithms are obtained by hybridizing with other evolutionary algorithms. These algorithms are named memetic optimization algorithms in the literature.

Thank you