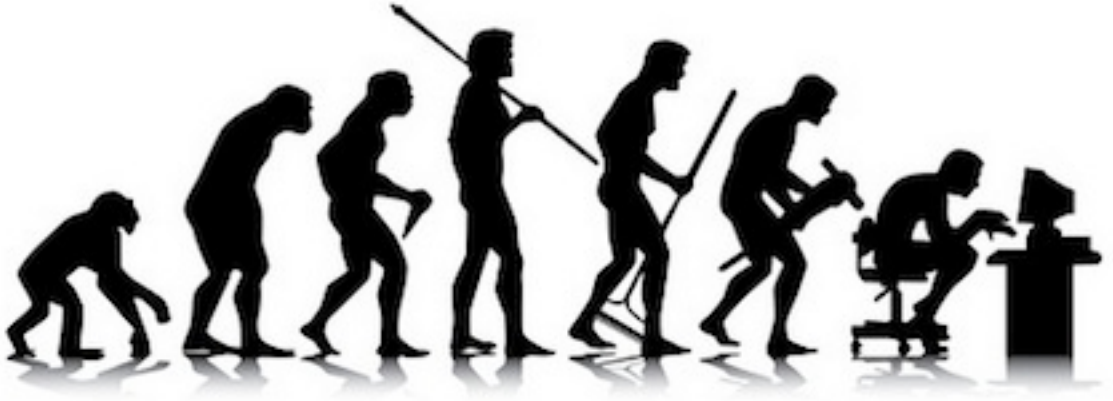


An Introduction to Genetic Algorithms: Method and Implementation

Dr. Anirban Mukhopadhyay

Professor

Department of Computer Science and Engineering

University of Kalyani

anirban@klyuniv.ac.in

www.anirbanm.in

Mathematical models in Optimization

- The general form of an *optimization problem*:

  min or *max* $f(x_1,\ldots,x_n)$ (objective function)

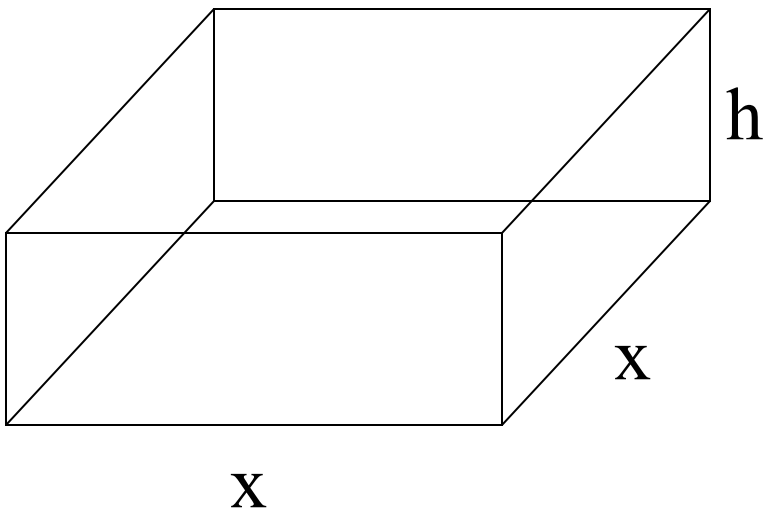
  subject to $g_i(x_1,\ldots,x_n) \geq 0$ (functional constraints)

  $x_1,\ldots,x_n \in S$ (set constraints)

- $x_1,\ldots,x_n$ are called decision variables
- In words,

  the goal is to find $x_1,\ldots,x_n$ that
  - satisfy the constraints;
  - achieve min (max) objective function value.

Ex. 1 A manufacturer wants to design an open box having a square base and a surface area of 108 in^2. What dimensions will produce a box with maximum volume?



Since the box has a square base, its volume is

$V = x^2h$

Note: We call this the primary equation because it gives a formula for the quantity we wish to optimize.

The surface area = the area of the base + the area of the 4 sides.

$S.A. = x^2 + 4xh = 108$

We want to maximize the volume, so express it as a function of just one variable. To do this, solve $x^2 + 4xh = 108$ for h.

$$h = \frac{108 - x^2}{4x}$$ Substitute this into the Volume equation.

$$V = x^2 h = x^2\left(\frac{108 - x^2}{4x}\right) = 27x - \frac{x^3}{4}$$

To maximize V we find the derivative and make it equal to 0.

$$\frac{dV}{dx} = 27 - \frac{3x^2}{4} = 0 \qquad 3x^2 = 108 \qquad i.e., \quad x = \pm 6$$

Hence $$h = \frac{108 - 36}{24} = 3$$

We can conclude that V is a maximum when x = 6 and the dimensions of the box are **6 in. x 6 in. x 3 in.**

Very Simple, But …

- Derivative-based methods do not work always.
  - When derivatives do not exist.
  - When the objective function cannot be expressed as a mathematical function of decision variables.
  - The optimization problem is discrete.
  - There are large number of decision variables and partial derivatives of the objective function yield a large number of equations to be solved.

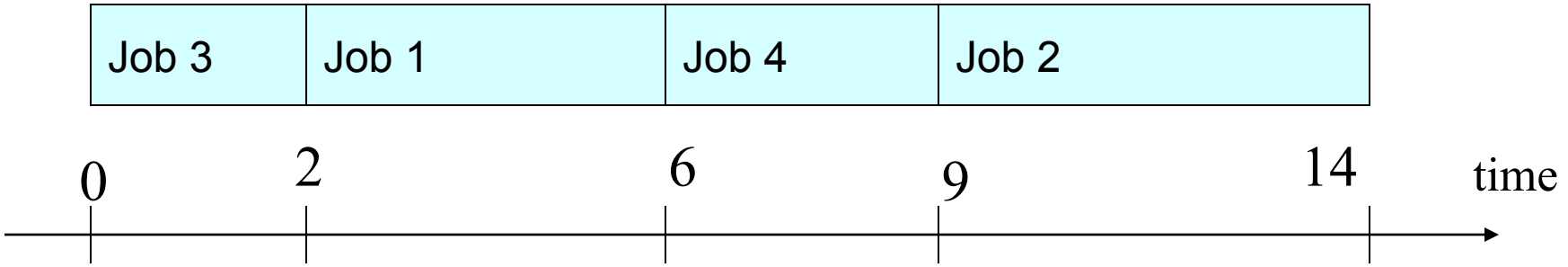
Examples of Optimization Problem not Solvable by Discrete optimization

Job Scheduling

- There are 4 jobs that should be processed on the same machine. (*Can't be processed simultaneously*).

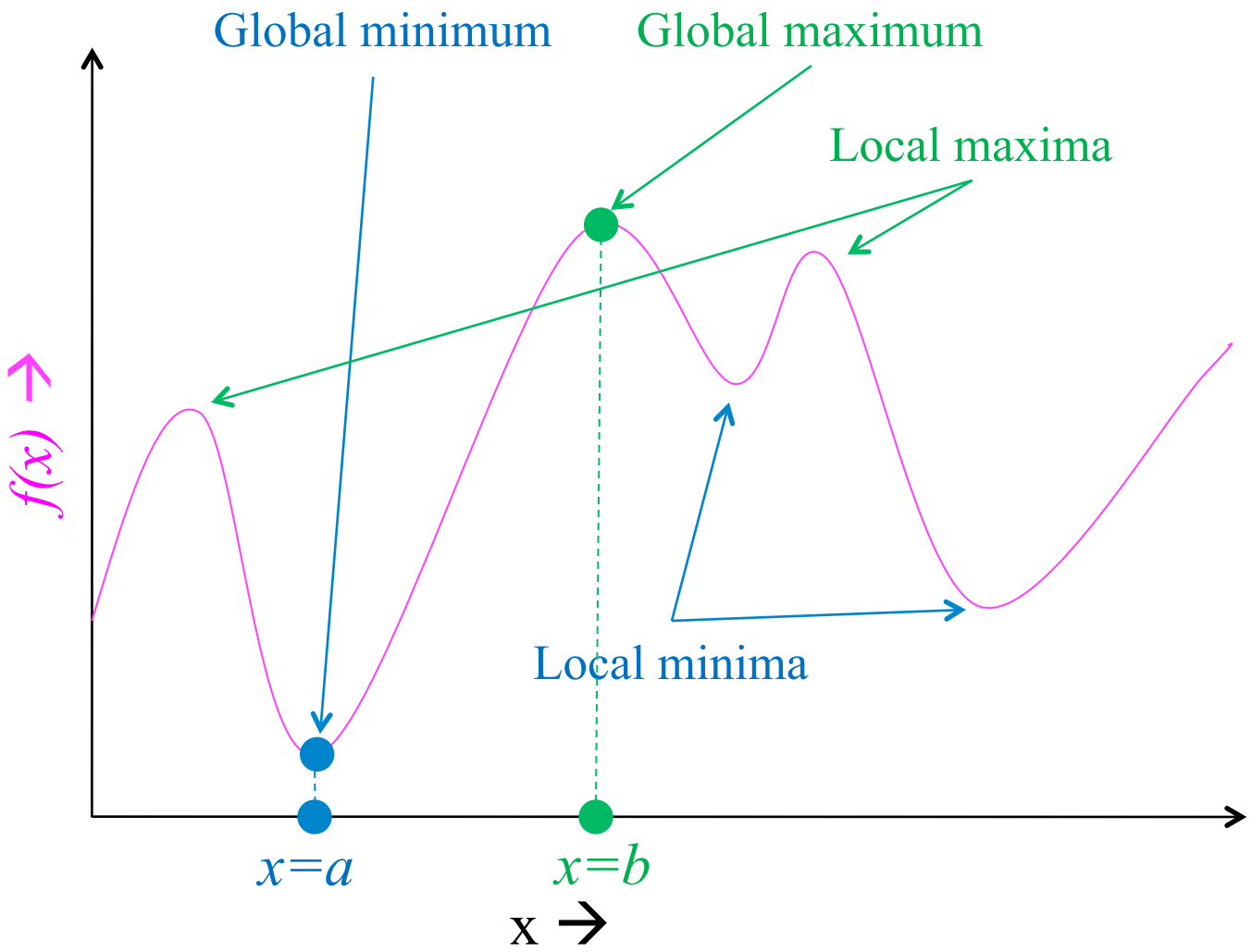
  Job *k* has processing time p_k .

  Here is an example of a possible schedule:



- Goal: Find a schedule which minimizes the average completion time of the jobs.

Local vs Global Optima



Some Metaheuristic Optimization Tools

- Genetic Algorithms
- Differential Evolution

- Particle Swarm Optimization
- Ant Colony Optimization

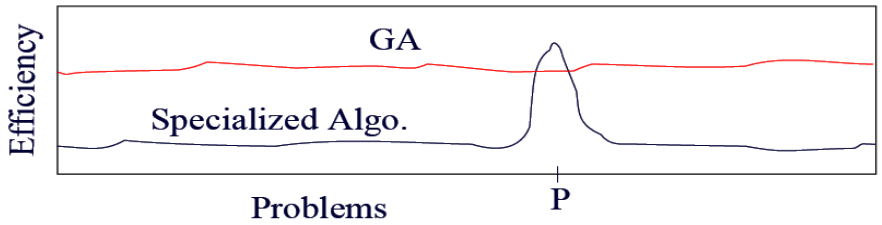
- Simulated Annealing

Genetic Algorithms (GA)

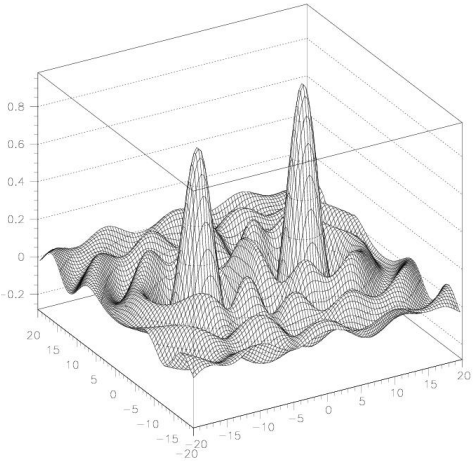
- Randomized search and optimization technique guided by the principle of natural genetic systems.
- Inspired by the biological evolution process
- Uses concepts of "Natural Selection", "Genetic Inheritance" and "Survival of the Fittest" (Darwin 1859)
- Originally developed by **John Holland** (1975) and gained popularity during late 80's.

Why GA?

- Most real life problems can not be solved in polynomial amount of time using any deterministic algorithm.
- Sometimes near optimal solutions that can be generated quickly are more desirable than optimal solutions which require huge amount of time.
- When the problem can be modeled as an optimization one.
- Efficiently searches for global optima when multiple optima exist.
- Parallel implementation is easy.



Specialized algorithms – best performance for special problems
Genetic algorithms – good performance over a wide range of problems



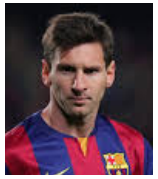
GA Features

- Evolutionary Search and Optimization Technique
- Principles of Evolution (survival of the fittest and inheritance).
- Work with encoding of the parameter set.
- Searches from a population of points.
- Uses probabilistic transition rules.

GA vs. Nature

• **A solution (phenotype)**	**Individual**
• Representation of a solution (genotype)	Chromosome
• **Components of the representation**	**Genes**
• Solution's quality (fitness function)	Individual's degree of ability to adopt with surrounding
• **Stochastic operators**	**Selection, Crossover Mutation**

Genotype vs. Phenotype

Population of Individuals		Fitness
Genotype	*Phenotype*	Performance Index
		23.5
		26.4
		20.3
		21.7

Simple GA

Produce an initial population of individuals

Evaluate the fitness of all individuals

While termination condition not met **do**

Select fitter individuals for reproduction

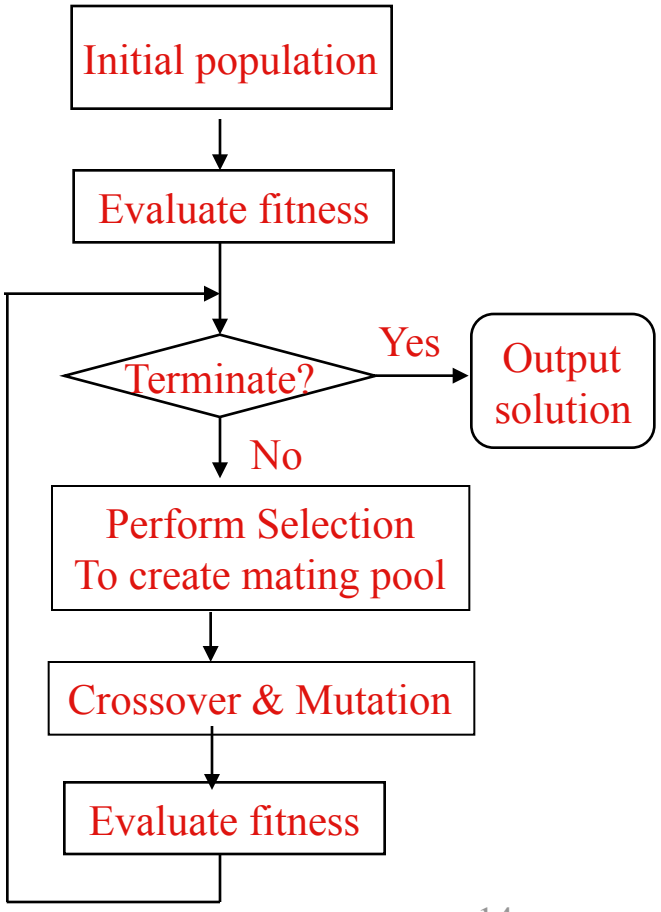
Recombine (crossover) between individuals

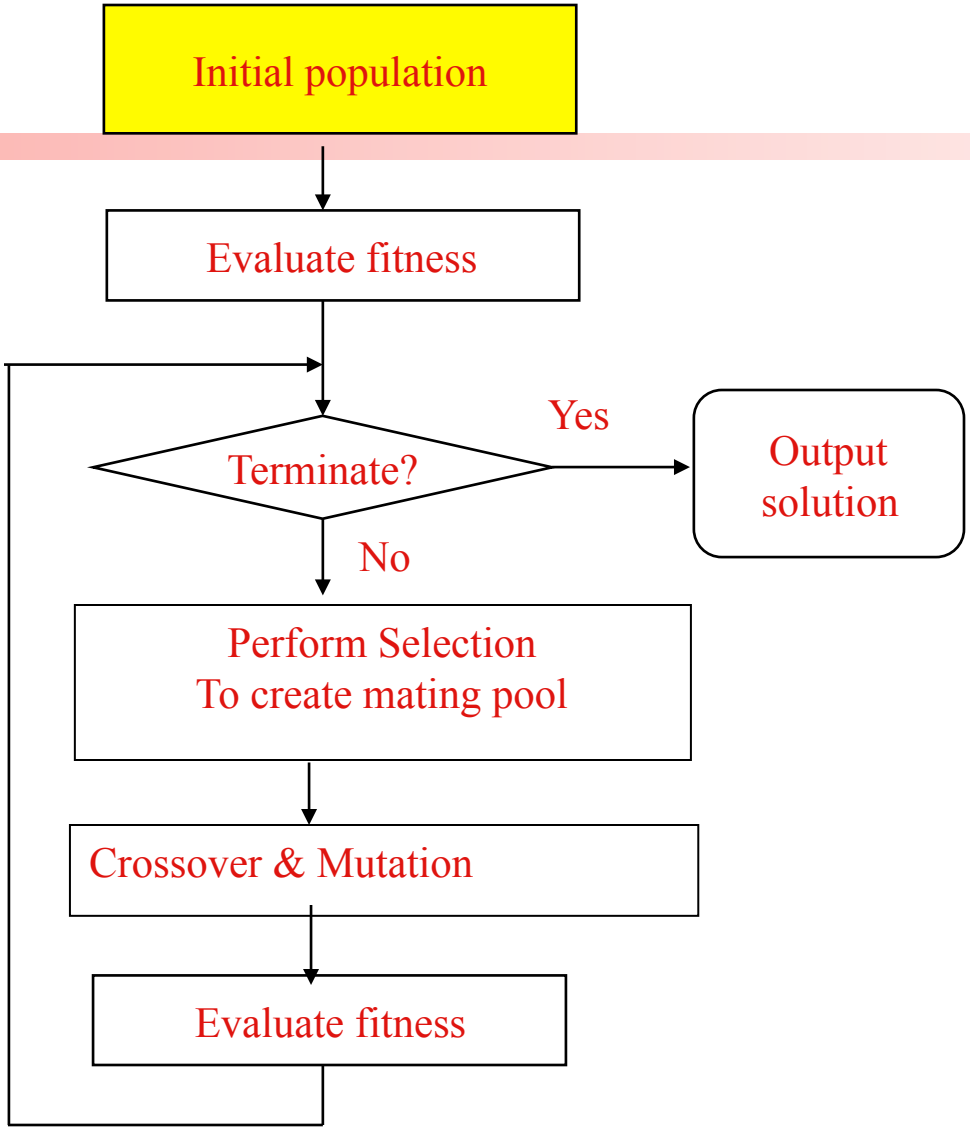
Mutate individuals

Evaluate the fitness of the modified individuals

Generate a new population

End while




Initial population
Evaluate fitness
Terminate?
Yes
Output solution
No
Perform Selection
To create mating pool
Crossover & Mutation
Evaluate fitness

Encoding and Population

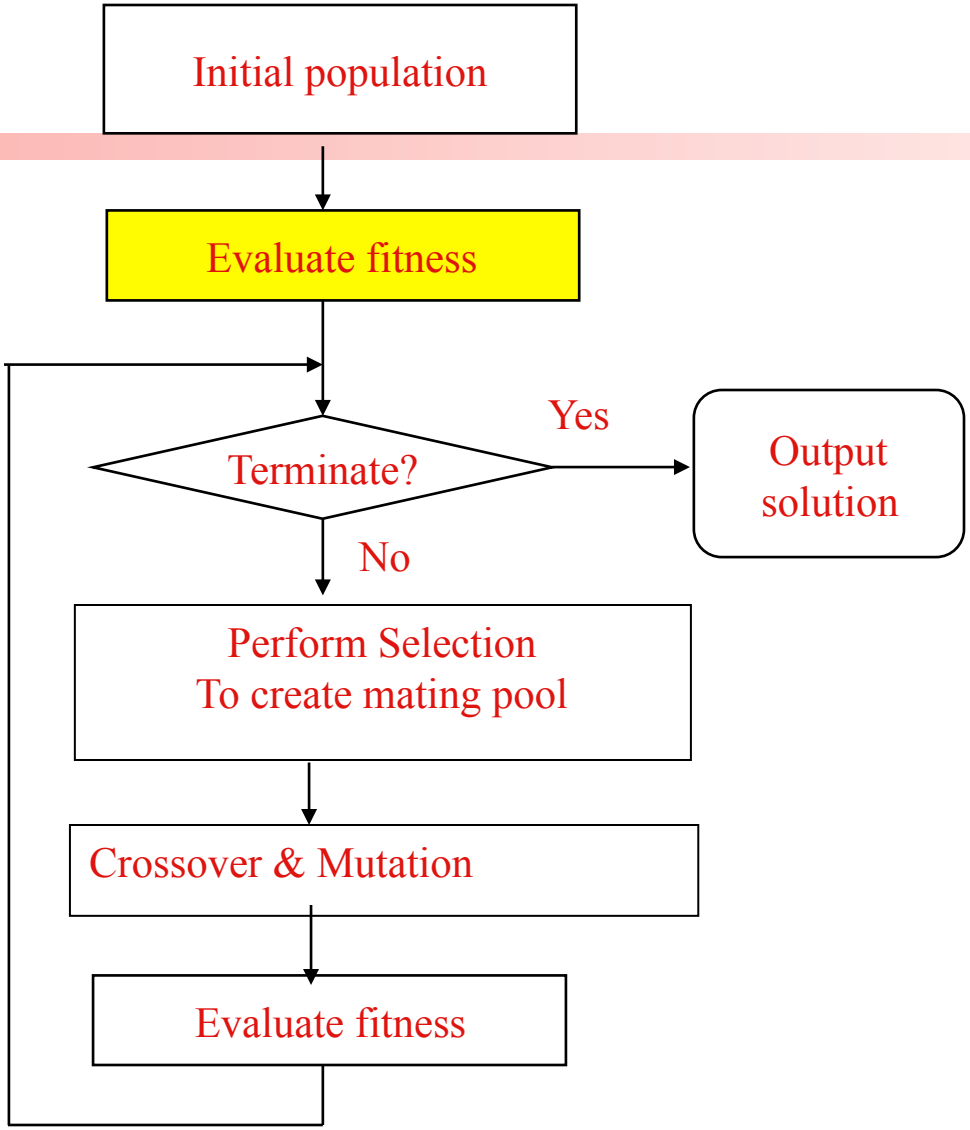
- **Chromosome encodes a solution in the search space**
  - Usually as strings of 0's and 1's
  - If l is the string length, number of different chromosomes (or strings) is 2^l
- **Population**
  - A set of chromosomes in a generation
  - Population size is usually constant
  - Common practice is to choose the initial population randomly.

Population Initialization Sample C Code

```
#include<stdlib.h>
…
…
for(int i=0;i<P;i++)  // loop of i to choose chromes from [0] to [P-1]
{
  for(int j=0;j<l;j++)  // loop of j to choose the gene of the chrom from [0] to [l-1]
 {
   int randn=rand();  // creating random value  between 0 and RAND_MAX-1
   randn=(randn%2);  // make the random value 0 or 1 only
   popcurrent[i][j] =randn; // initializing the bit[j] of chrom[i] with random
  }
}
```

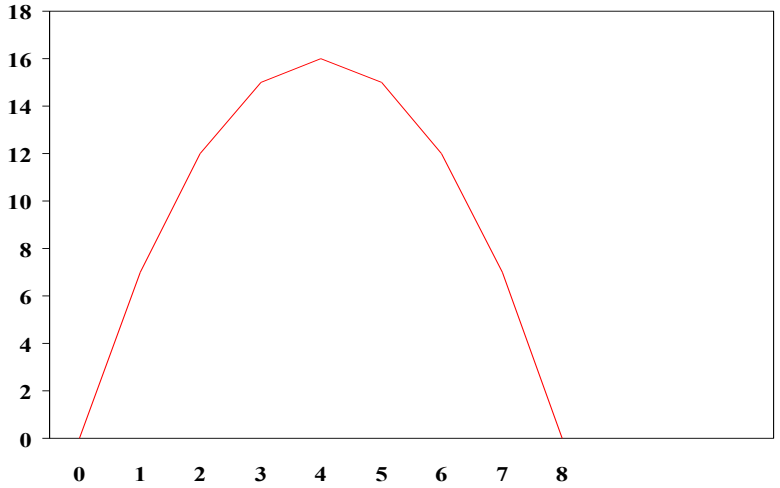
Population Initialization
Sample Matlab Code

```
popcurrent = rand(P,l); % Generate P x l matrix with random values between 0 & 1
popcurrent = round(popcurrent); % Convert popcurrent to binary matrix
```


Initial population
Evaluate fitness
Terminate?
Yes
Output solution
No
Perform Selection
To create mating pool
Crossover & Mutation
Evaluate fitness

Encoding and Population - Example

Optimize $f(x) = x(8 - x)$, $x=[0,8]$



Binary String of 8 bits

0-255 ⟷ 0-8

1	0	0	1	1	0	1	0

= 154

$x = 0 + (8/255) * 154 = 4.8313$

Fitness Calculation of a Chromosome (C Code)

```
int dec=0, j=0;
for(int i=l-1;i>=0;i--)  // Binary to decimal conversion
{
   dec+=(chrom[i]*pow(2,j));
   j++;
}

x= (8/255)*dec;  // Bringing the range of dec within 0 to 8
fit = x*(8-x);   // Computing fitness value
```

Fitness Calculation of a Chromosome (Matlab Code)

```
chromstr = num2str(chrom);  % Convert chromosome into string
dec=bin2dec(chromstr); % Binary to decimal conversion

x= (8/255)*dec;  % Bringing the range of dec within 0 to 8
fit = x*(8-x);   % Computing fitness value
```

Fitness Evaluation

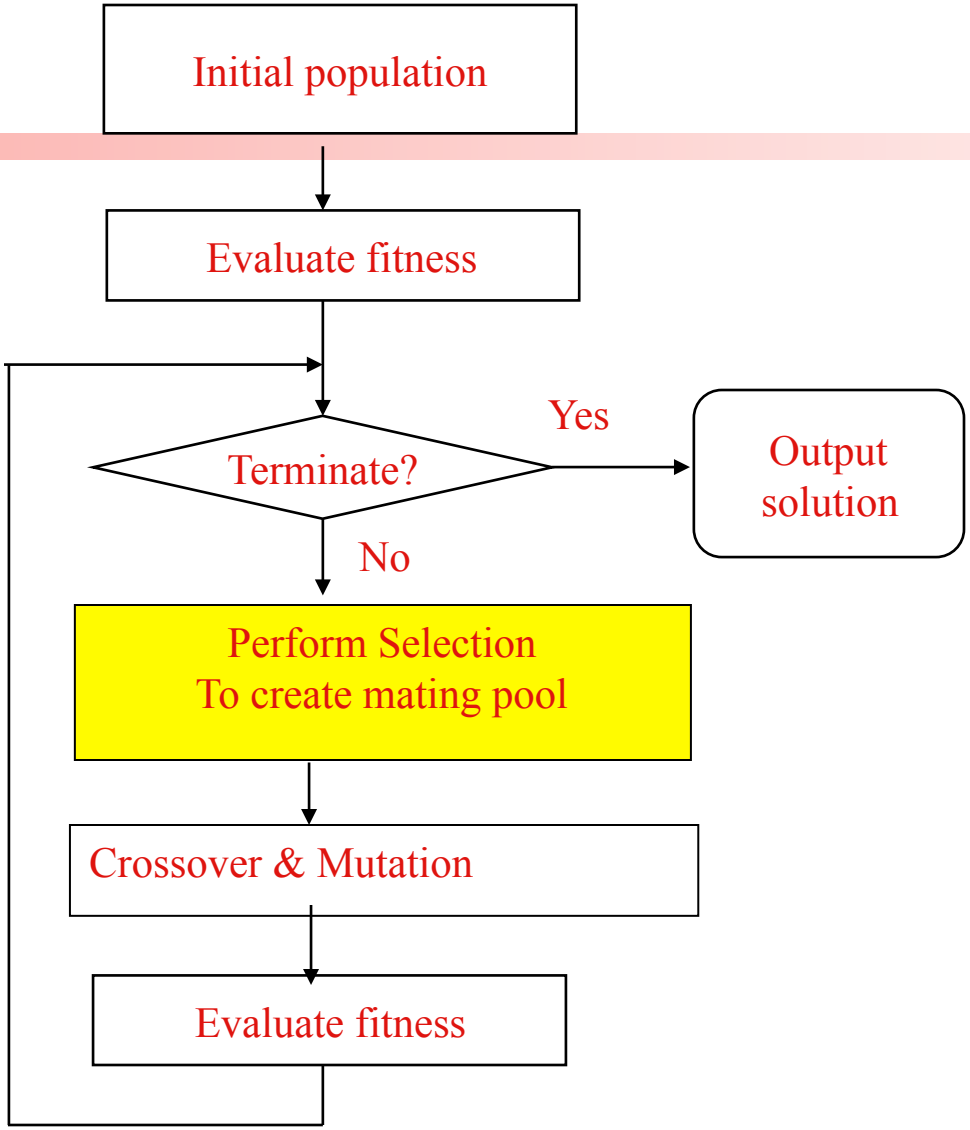
- Fitness/objective function is associated with each chromosome
- This indicates the degree of goodness of the encoded solution
- The only problem specific information (also known as the payoff information) that GA uses
- If minimization problem is to be solved then

  fitness = 1/objective.

Fitness Evaluation - Example

Function *f(x) = x(8-x)*

Population (size = 4)	Corresponding *x*	Fitness/ Objective Fn.
1 0 01 1 0 1 0	4.8313	15.3089
0 1 1 0 0 1 1 1	3.2313	15.4091
0 0 0 1 0 1 0 1	0.6588	4.8363
1 0 1 1 1 1 0 0	5.8980	12.3975


Initial population
Evaluate fitness
Terminate?
Yes
Output solution
No
Perform Selection
To create mating pool
Crossover & Mutation
Evaluate fitness

Selection

- More copies to good strings
- Fewer copies to bad string

- Proportional selection scheme
  - Number of copies taken to be directly proportional to its fitness
  - Mimics the natural selection procedure to some extent.

- **Roulette wheel selection** and **Tournament selection** are two frequently used selection procedures.

Tournament - Selection

- Repeat until mating pool is full
  - Select a set (size < population size) of chromosomes randomly.
  - Copy the best chromosome among them into the mating pool.

- Usually tournament size is 2 (**binary tournament**).

- The chromosome with lowest fitness value can never be copied into the mating pool.

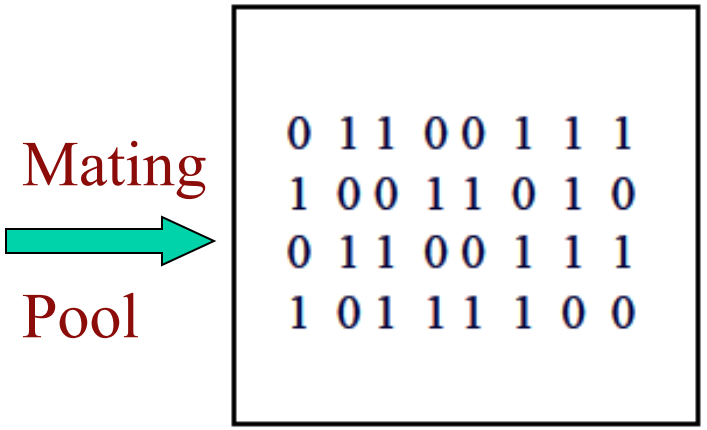
Binary Tournament Selection - Example

Chromosome #	Fitness
1	15.3089
2	15.4091
3	4.8363
4	12.3975

Tournaments

Round 1: Chrom 2 vs. 4
Round 2: Chrom 1 vs. 3
Round 3: Chrom 2 vs. 3
Round 4: Chrom 3 vs. 4

- Round 1: Chromosome 2 is selected
- Round 2: Chromosome 1 is selected
- Round 3: Chromosome 2 is selected
- Round 4: Chromosome 4 is selected



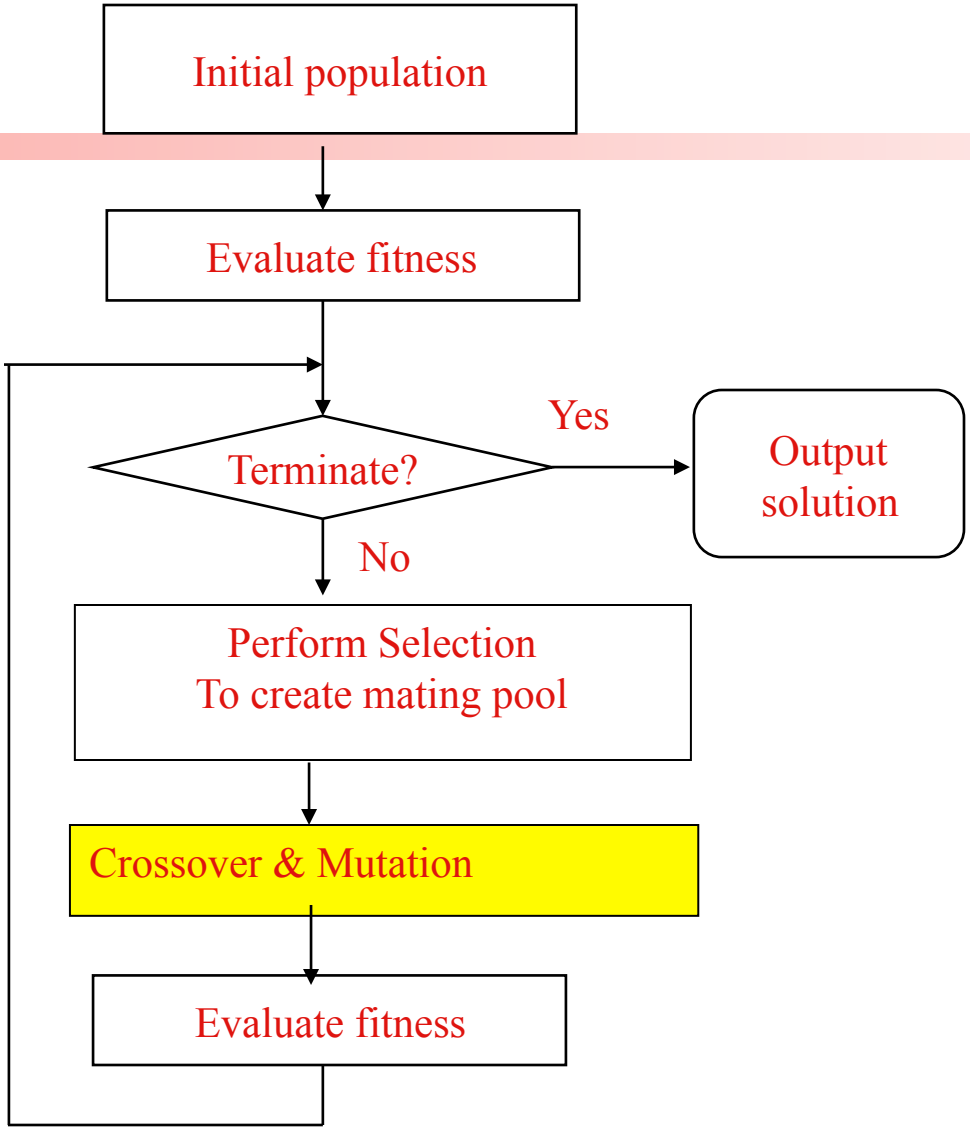
Binary Tournament in C

```
for(i=0;i<P;i++){
//Pick two random candidates between 0 and P-1
   int cand1 = rand()%P;
   int cand2 = rand()%P;
   if (fit[cand1] >= fit[cand2]){  //Compare to find better chromosome
            selected = cand1;}
   else{
            selected = cand2;}

   for (j=0;j<l;j++) // Insert better in mating pool
            matpool[i][j]=popcurrent[selected][j];
}
```

Binary Tournament in Matlab

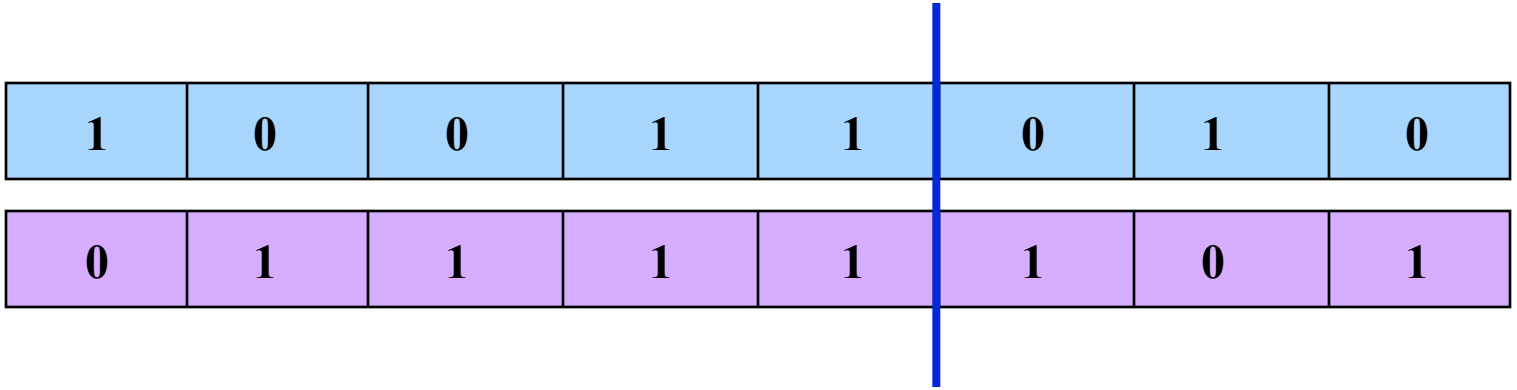
```
for i=1:P
% Pick two random candidates between 0 and P-1
   cand1 = round(unifrnd(1,P,1,1));
   cand2 = round(unifrnd(1,P,1,1));
   if fit(cand1) >= fit(cand2)  % Compare to find better chromosome
            selected = cand1;
   else
            selected = cand2;
   end
   matpool(i,:) = popcurrent(selected,:); % Insert better in mating pool
end
```


Initial population
Evaluate fitness
Terminate?
Yes
Output solution
No
Perform Selection
To create mating pool
Crossover & Mutation
Evaluate fitness

Crossover

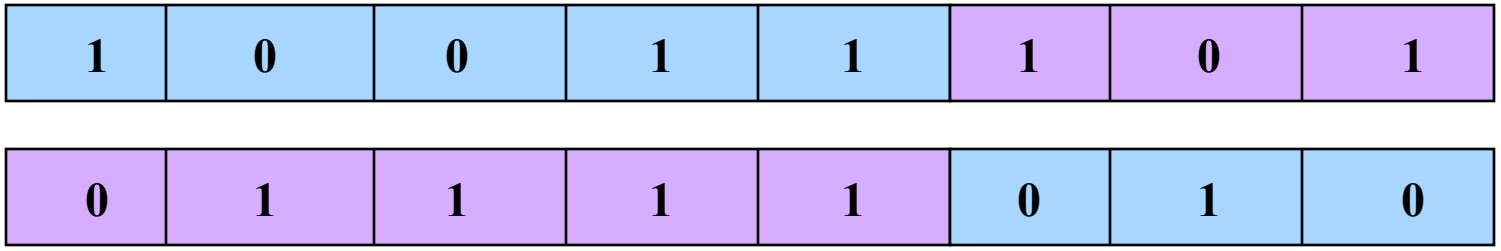
- Exchange of genetic information
- It takes place between randomly selected parent chromosomes
- **Single point crossover** and **Uniform crossover** are the most commonly used schemes.
- Probabilistic operation

Single Point Crossover – Example



Here l (string length) = 8. Let k (crossover point) = 5

Offspring formed:



Single Point Crossover in C

```
int cp = rand()%l; //Generate random crossover point between 0 and l-1

for(i=0;i<l;i++)
{
    if(i<=cp){ //No change upto crossover point
        child1[i]=parent1[i];
        child2[i]=parent2[i];
   }else{  // Exchange beyond crossover point
        child1[i]=parent2[i];
        child2[i]=parent1[i];
   }
}
```

Single Point Crossover in Matlab

```
% Generate random crossover point between 0 and l-1
cp = round(unifrnd(1,l,1,1));

% Create child1 with first part of parent1 and second part of parent2
child1=[parent1(1:cp), parent2(cp+1:end)];

% Create child2 with first part of parent2 and second part of parent1
child2=[parent2(1:cp), parent1(cp+1:end)];
```

Uniform Crossover – Example

Parents:

1	0	0	1	1	0	1	0
0	1	1	1	1	1	0	1

Mask:

1	1	0	1	0	1	1	0

Offspring formed:

0	1	0	1	1	1	0	0
1	0	1	1	1	0	1	1

Uniform Crossover in C

```
for(i=0;i<l;i++)
{
 int mask=rand()%2;
    if(mask==0){ //No change if mask bit is 0
        child1[i]=parent1[i];
        child2[i]=parent2[i];
   }else{  // Exchange if mask bit is 1
        child1[i]=parent2[i];
        child2[i]=parent1[i];
   }
}
```

Uniform Crossover in Matlab

```
child1=parent1; % Copy parent1 into child1
child2=parent2; % Copy parent2 into chold2

mask = round(rand(1,l)); % Generate binary mask vector of length l

ind = find(mask==1); % Find indices where mask bits are 1

% Exchange parents where mask bits are 1
child1(ind)=parent2(ind);
child2(ind)=parent1(ind);
```

Crossover Operation Loop

Input: *Mating_Pool, Crossover Probability* (μ_c)

Output: *Next_PopX*

While *Next_PopX* is not filled up ***do***

>> Choose two random chromosomes from *Mating_Pool* as parents;

If crossover is to be done ***then***

>> Perform crossover on parents;

>> Put the offspring solutions in *Next_PopX*;

Else

>> Put the parents in *Next_PopX* directly;

EndIf

EndWhile

Crossover Operation Loop

Input: *Mating_Pool, Crossover Probability* (μ_c)

Output: *Next_PopX*

While *Next_PopX* is not filled up ***do***

>> Choose two random chromosomes from *Mating_Pool* as parents;

If crossover is to be done ***then***

>> Perform crossover on parents;

>> Put the offspring solutions in *Next_PopX*;

Else

>> Put the parents in *Next_PopX* directly;

EndIf

EndWhile

Crossover Operation Loop

Input: *Mating_Pool, Crossover Probability* (μ_c)

Output: *Next_PopX*

While *Next_PopX* is not filled up ***do***

>> Choose two random chromosomes from *Mating_Pool* as parents;

>> r = A random number in (0,1);

If $r < \mu_c$ ***then***

>> Perform crossover on parents;

>> Put the offspring solutions in *Next_PopX*;

Else

>> Put the parents in *Next_PopX* directly;

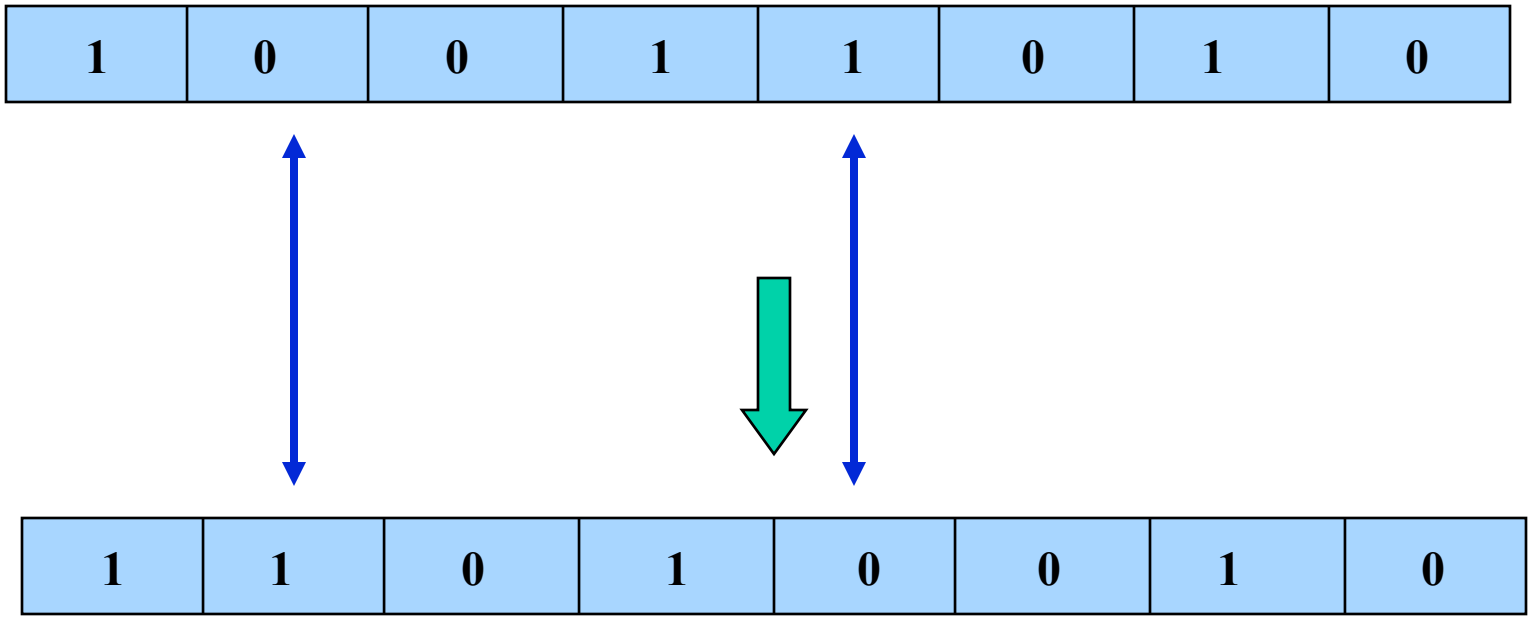
EndIf

EndWhile

Mutation

- **Random alteration** in the genetic structure
- Introduces **genetic diversity** into the population.
- Exploration of new search areas
- Mutating a binary gene involves simple **negation of the bit**
- Mutating a real coded gene defined in a variety of ways
- Probabilistic operation

Mutation – Example



Mutation Operation Loop

Input: *Next_PopX*, *Mutation Probability* (μ_m)

Output: *Next_PopM*

For for each chromosome in Next_PopX ***do***

  For each bit b_i in the chromosome ***do***

    If the bit b_i is to be mutated ***then***

      $>> b_i = 1 - b_i$;

    EndIf

  EndFor

Endfor

Mutation Operation Loop

Input: *Next_PopX*, *Mutation Probability* (μ_m)

Output: *Next_PopM*

For for each chromosome in Next_PopX ***do***

  For each bit b_i in the chromosome ***do***

    If the bit b_i is to be mutated ***then***

      >> $b_i = 1 - b_i$;

    EndIf

  EndFor

Endfor

Mutation Operation Loop

Input: *Next_PopX*, *Mutation Probability* (μ_m)

Output: *Next_PopM*

For for each chromosome in Next_PopX ***do***

For each bit b_i in the chromosome ***do***

>> r = A random number in (0,1);

If $r < \mu_m$ ***then***

>> $b_i = 1 - b_i$;

EndIf

EndFor

Endfor

Bit-flip Mutation in C

```
for(i=0;i<P;i++)  // For all chromosomes in population
{
   for(j=0;j<l;j++)  // For each bit of the chromosome
   {
      r = randrange(0,1);  // Generate random number between 0 and 1
      if(r<mu_prob) // Checks whether the bit is to be mutated
          pop[i][j]=1-pop[i][j];  // Flip the bit
   }
}
```

```
double randrange(double low, double high) {
d = ((double)rand() * ( high - low ) ) / (double)RAND_MAX + low;
return(d);
}
```

Bit-flip Mutation in Matlab

```
for i=1:P  // For all chromosomes in population
    for j=1:l  // For each bit of the chromosome
      r=rand;  // Generate random number between 0 and 1
      if r<mu_prob // Checks whether the bit is to be mutated
          pop(i,j)=1-pop(i,j);  // Flip the bit
      end
    end
end
```

Parameters

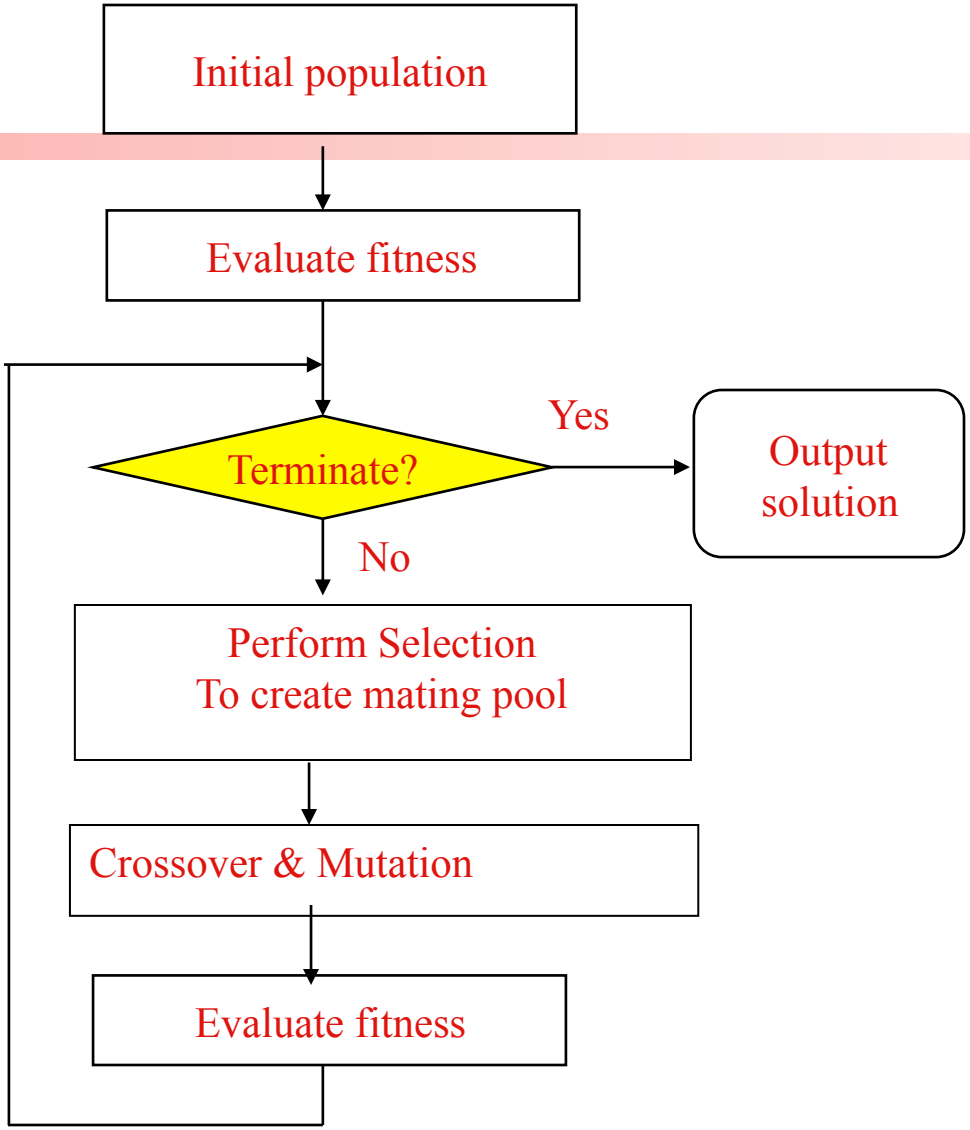
- Population size – usually fixed
- String length – usually fixed
- Probabilities of crossover, μ_c, and mutation, μ_m

  μ_c is kept high (~ 1.0) and μ_m is kept low (~ 0.0).
- Termination criteria
- Parameters are often manually tuned
- Sometimes may be adaptive.

Parameters – Example

For the example being considered,

- P = 4, l = 8.
- But for most realistic cases P is usually chosen in the range 50-100.
- μ_c = [0.6-0.9],
- μ_m = [0.01-0.1].
- l usually depends on the required precision


Initial population
Evaluate fitness
Terminate?
Yes
Output solution
No
Perform Selection
To create mating pool
Crossover & Mutation
Evaluate fitness

Termination Criterion

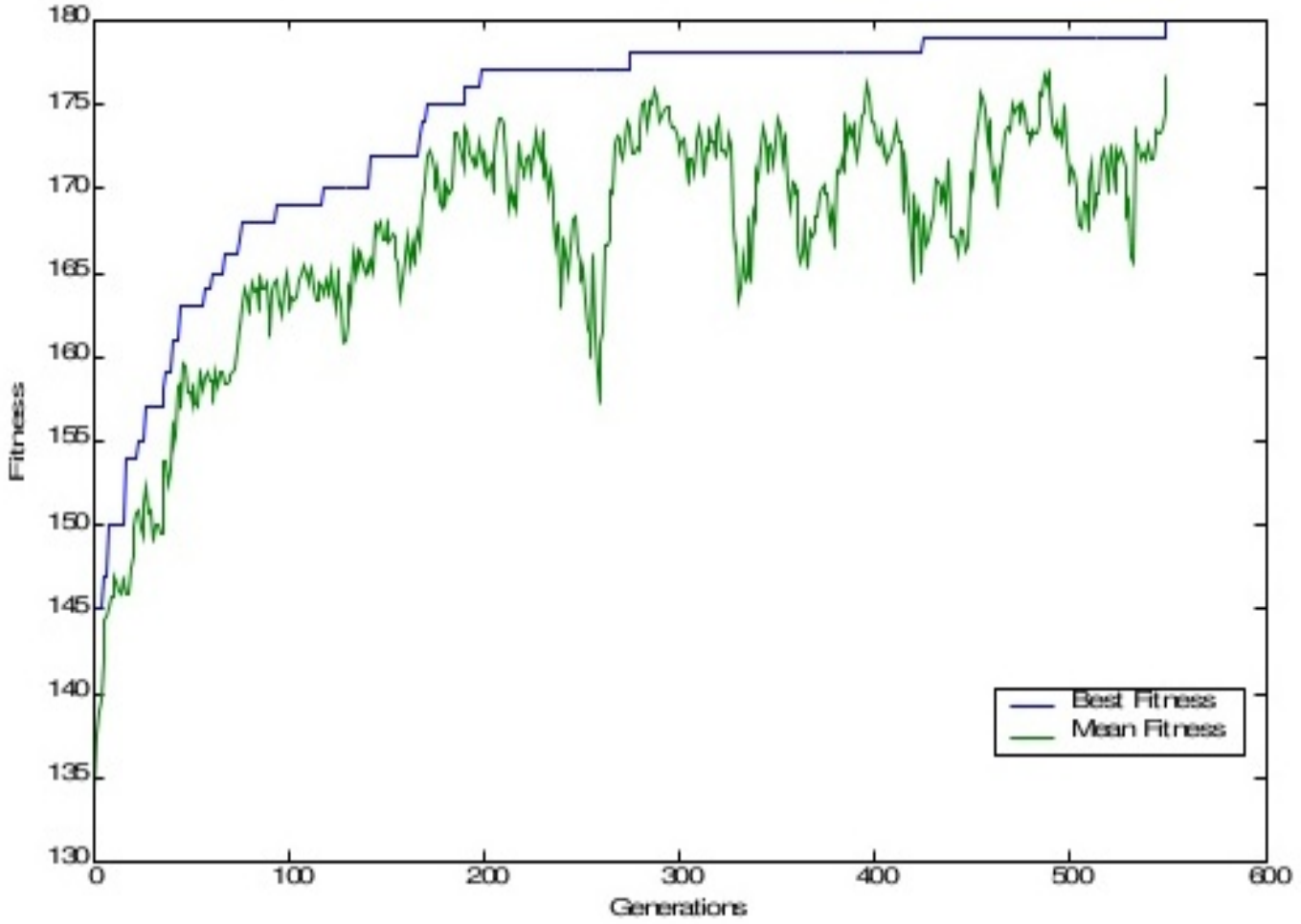
The cycle of selection, crossover and mutation are repeated for number of times until one of these occurs

- Average fitness value of a population more or less constant over several generations,
- Desired objective function value is attained by at least one string in the population,
- **Number of generations (or iterations) is greater than some threshold ----- most commonly used.**
- **The best chromosome of the last generation is the solution**

Elitism

- The best string seen up to the current generation is preserved in a location either inside or outside the population.
- The best few chromosomes of generations *i* is copied to generation *i+1* directly.
- The remaining positions of generation *i+1* is filled up by selection, crossover and mutation.
- **Goal** – Not to loose the best solution obtained so far due to randomness.

Variation of Fitness over Generations



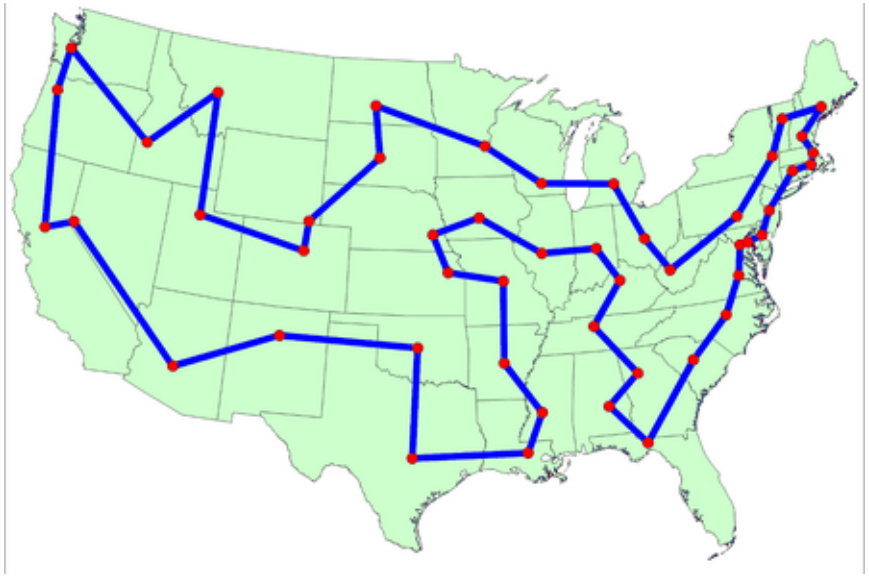
Solving Two Famous Problems using GA

- Traveling Salesman Problem
- 0-1 Knapsack Problem

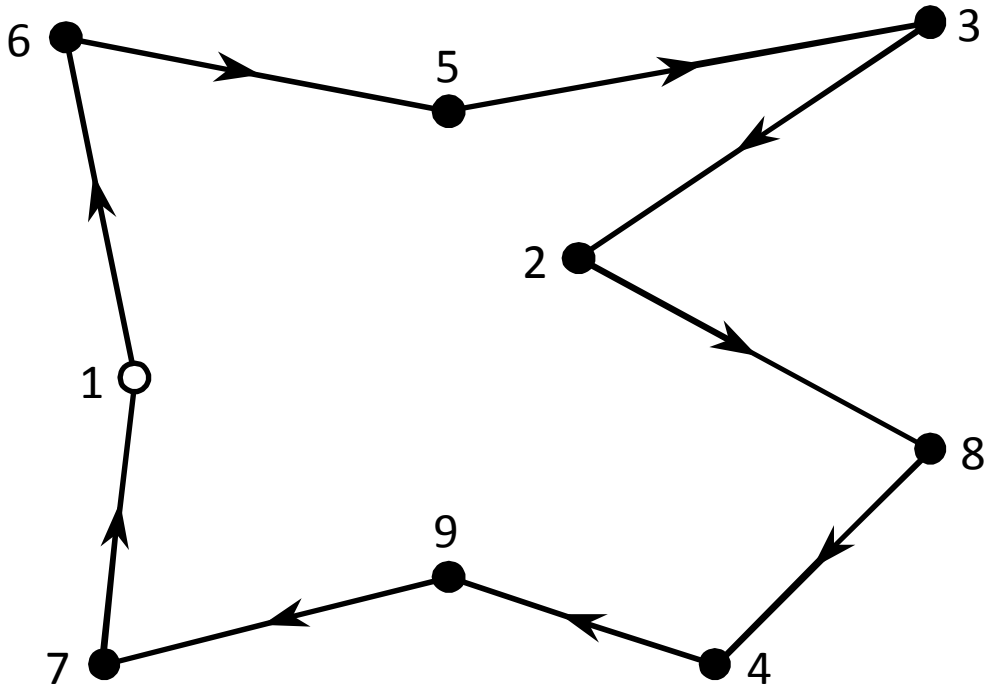
Another Example: The Traveling Salesman Problem (TSP)

- The traveling salesman must visit every city in his territory exactly once and then return to the starting point.
- The cost of travel between each pair of cities is given.
- Problem: How should he plan his itinerary for minimum total cost of the entire tour?

TSP is a hard problem



Tour Representation Example



1	6	5	3	2	8	4	9	7

TSP (Representation, Evaluation, Initialization and Selection)

Chromosome: A vector $v = (i_1\, i_{2\ldots}\, i_n)$ represents a tour (v is a permutation of {1,2,…,n}).

Example chromosome: 2 5 6 8 1 7 3 4

How to generate a random chromosome ?

Fitness function: Fitness f of a solution is the inverse cost of the corresponding tour.

Example: f= cost(2,5)+cost(5,6)+…cost(3,4)+cost(4,2)

Initialization: use either some heuristics, or a random sample of permutations of {1,2,…,n}

Selection: We shall use the roulette wheel selection

TSP (Crossover1)

Builds offspring by choosing a sub-sequence of a tour from one parent and preserving the relative order of cities from the other parent and feasibility

Example:

p_1 = (1 2 3 4 5 6 7 8 9) and

p_2 = (4 5 2 1 8 7 6 9 3)

First, the segments between cut points are copied into offspring

o_1 = (x x x 4 5 6 7 x x) and

o_2 = (x x x 1 8 7 6 x x)

TSP (Crossover2)

Next, starting from the second cut point of one parent, the cities from the other parent are copied in the same order

The sequence of the cities in the second parent is

9 – 3 – 4 – 5 – 2 – 1 – 8 – 7 – 6

After removal of cities from the first offspring we get

9 – 3 – 2 – 1 – 8

This sequence is placed in the first offspring

o_1 = (2 1 8 4 5 6 7 9 3), and similarly in the second

o_2 = (3 4 5 1 8 7 6 9 2)

TSP (Mutation: Inversion)

The sub-string between two randomly selected points in the path is reversed

Example:

(1 2 3 4 5 6 7 8 9) is changed into (1 2 7 6 5 4 3 8 9)

Such simple inversion guarantees that the resulting offspring is a legal tour

TSP (Mutation: Swap)

Two randomly selected cities in the path are swapped.

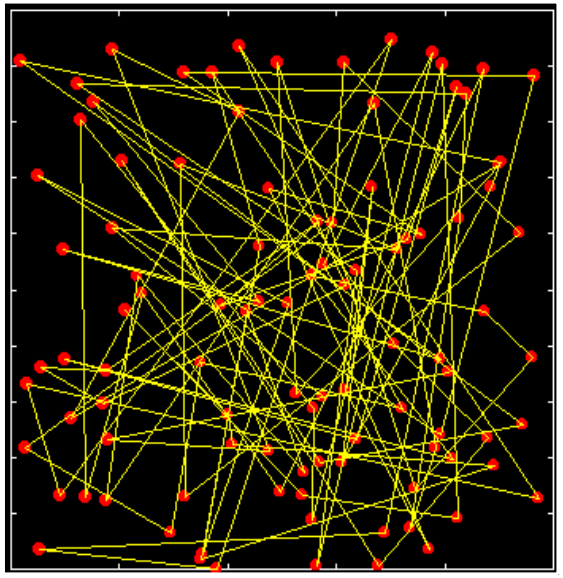
Example:

(1 2 3 4 5 6 7 8 9) is changed into (1 2 7 6 5 4 3 8 9)

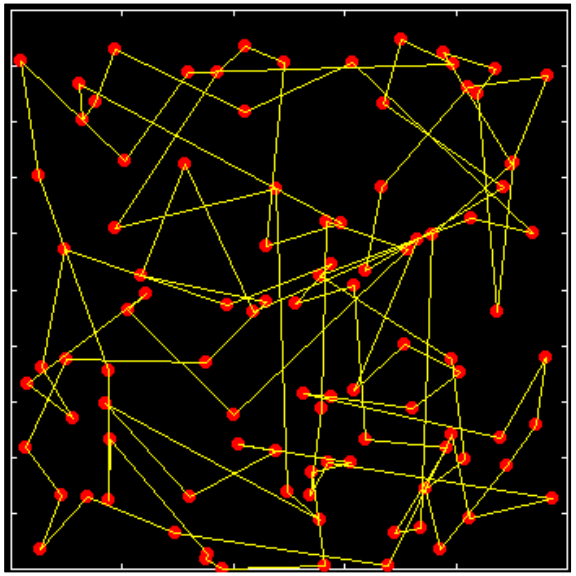
Such simple swap guarantees that the resulting offspring is a legal tour

TSP Example

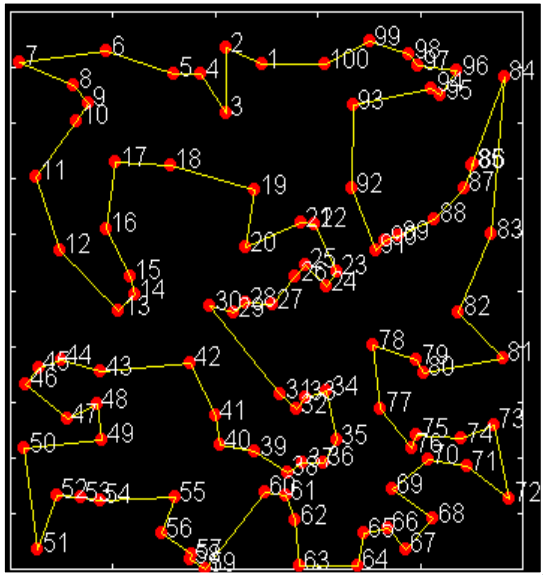
- A 100-city TSP using GA



Initial random solution



During GA process



Final solution

0-1 Knapsack Problem

Problem Description:

- You are going on a picnic.
- And have a number of items that you could take along.
- Each item has a weight and a benefit or value.
- You can take one of each item at most.
- There is a capacity limit on the weight you can carry.
- You should carry items with max. values.

Knapsack Problem

Example:

- **Item: 1 2 3 4 5 6 7**
- **Benefit: 5 8 3 2 7 9 4**
- **Weight: 7 8 4 10 4 6 4**
- **Knapsack holds a maximum of 22 kgs**
- **Fill it to get the maximum benefit**

Encoding Startegy

Start

- Encoding: 0 = not exist, 1 = exist in the Knapsack

  Chromosome: 1010110

Item.	1	2	3	4	5	6	7
Chro	1	0	1	0	1	1	0
Exist?	y	n	y	n	y	y	n

  => Items taken: 1, 3 , 5, 6.

- Generate random population of *n* chromosomes:

  a)0101010

  b)1100100

  c)0100011

Fitness Function

Maximum Weight = 22 kg

a) 0101010: Benefit= 19, Weight= 24 ✘

Item	1	2	3	4	5	6	7
Chro	0	1	0	1	0	1	0
Benefit	5	8	3	2	7	9	4
Weight	7	8	4	10	4	6	4

b) 1100100: Benefit= 20, Weight= 19. ✓

c) 0100011: Benefit= 21, Weight= 18. ✓

Constrained Optimization

Penalty Function Approach

Total Benefit of the selected items = B
Total Weight of the selected items = W
Maximum Weight = MW

If W<= MW
    Fitness = B
Else
    Fitness = B − K(W − MW)
End

Constrained Optimization

Binary Tournament Approach

Fitness = B

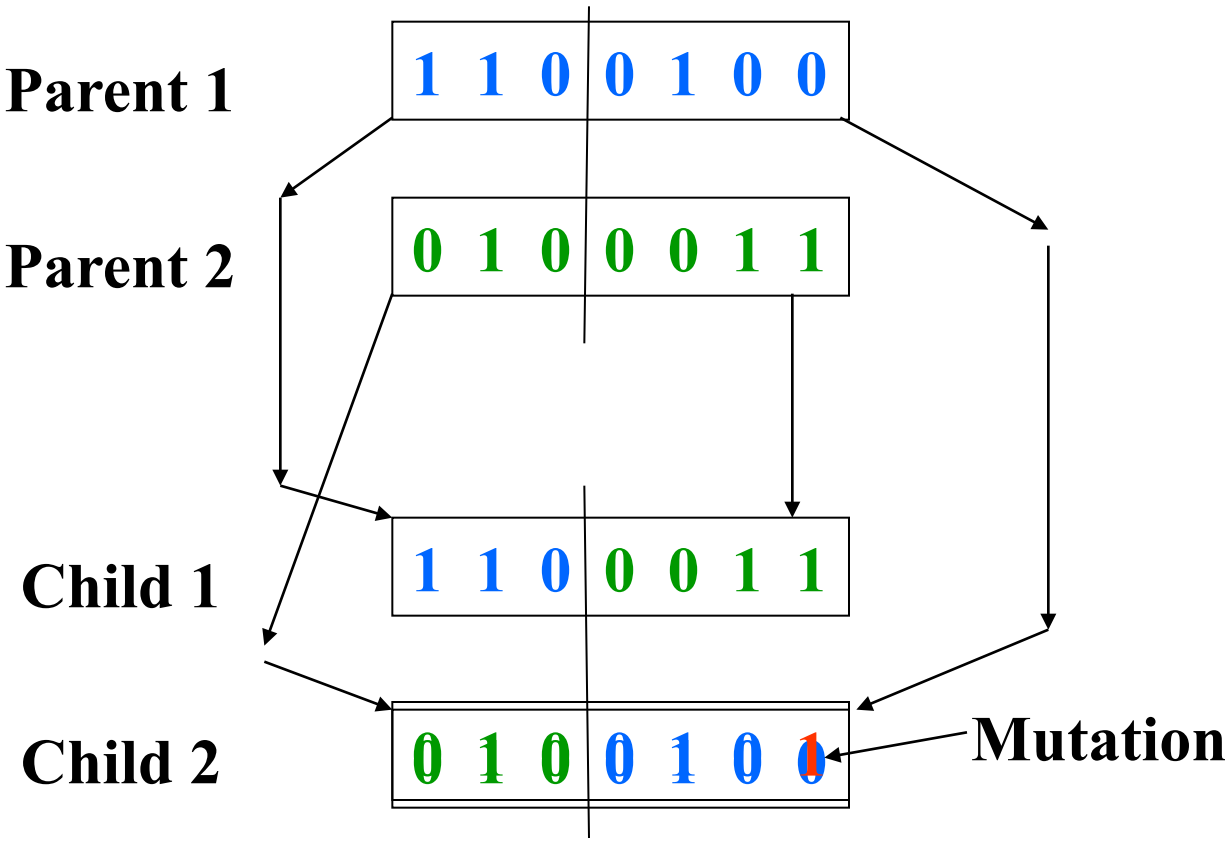
If both solutions are feasible
then select the one with larger B

If one solution is feasible and the other is infeasible
then select the feasible solution

If both solutions are infeasible
then select the solution with smaller W

Crossover and Mutation

Crossover & Mutation



Thank You

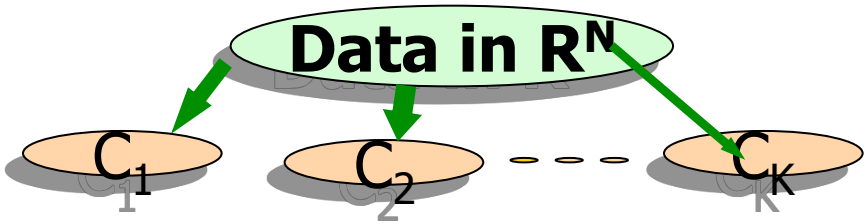
www.anirbanm.in

anirban@klyuniv.ac.in

GA based Clustering

Clustering

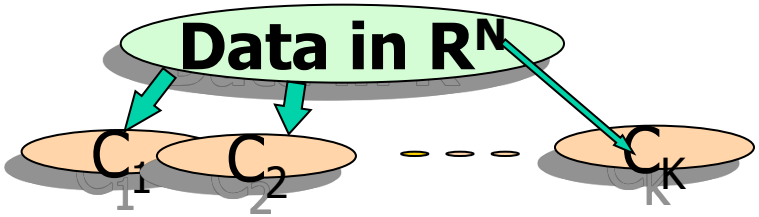
- Clustering is a popular unsupervised pattern classification technique which partitions the input space into *K* regions based on some similarity/dissimilarity measure.
  - The value of *K* may or may not be known a priori.



$C\_i \neq \emptyset$

$C\_i \cap C\_j = \emptyset$

Crisp clustering

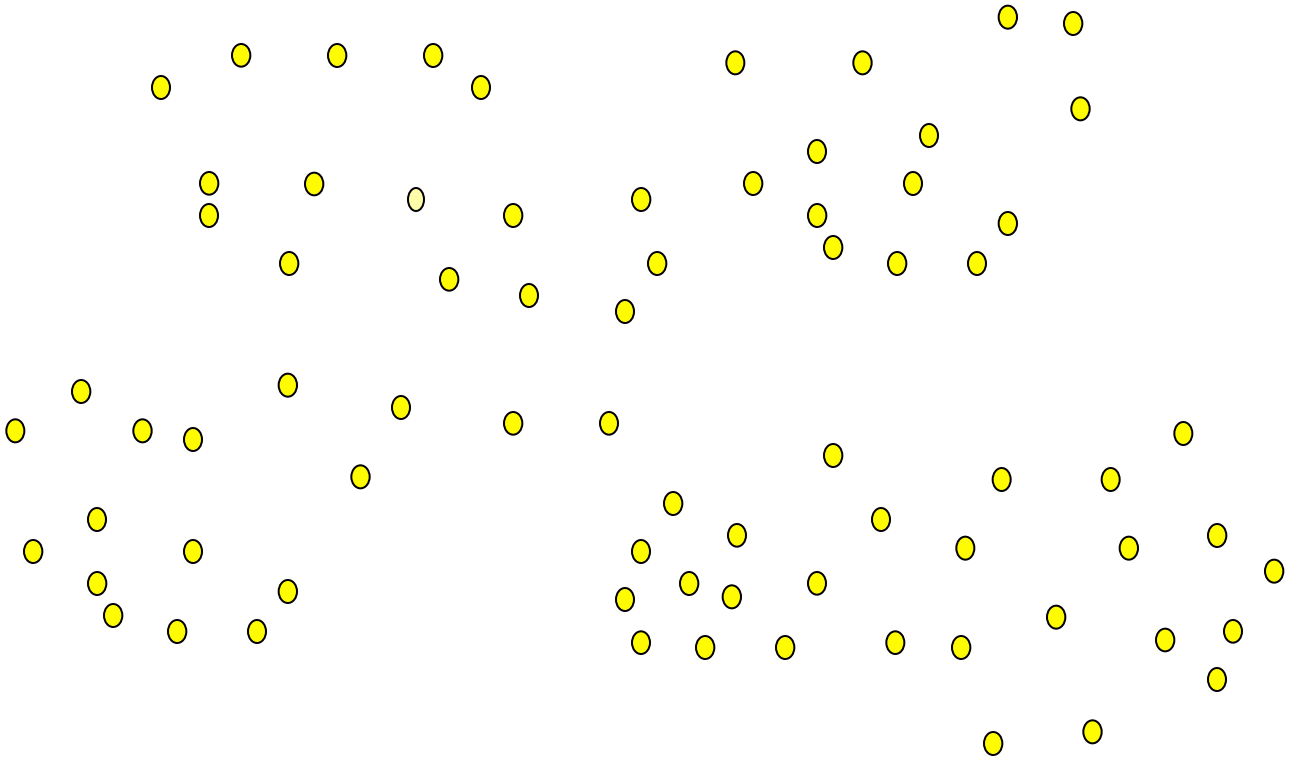


$C\_i \neq \emptyset$

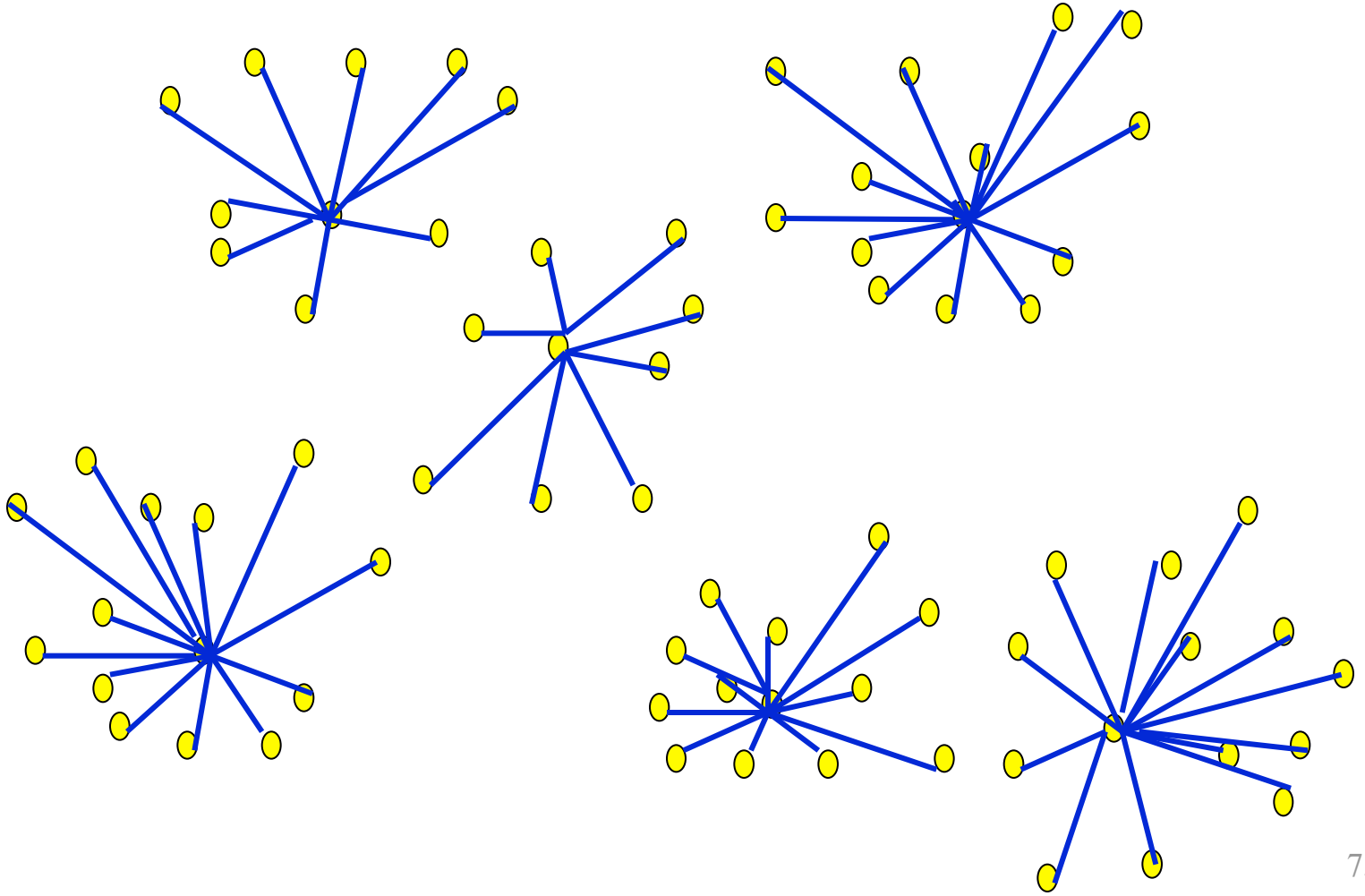
$C\_i \cap C\_j \neq \emptyset$

Fuzzy clustering

Spatial Clustering



Spatial Clustering



K-means Clustering

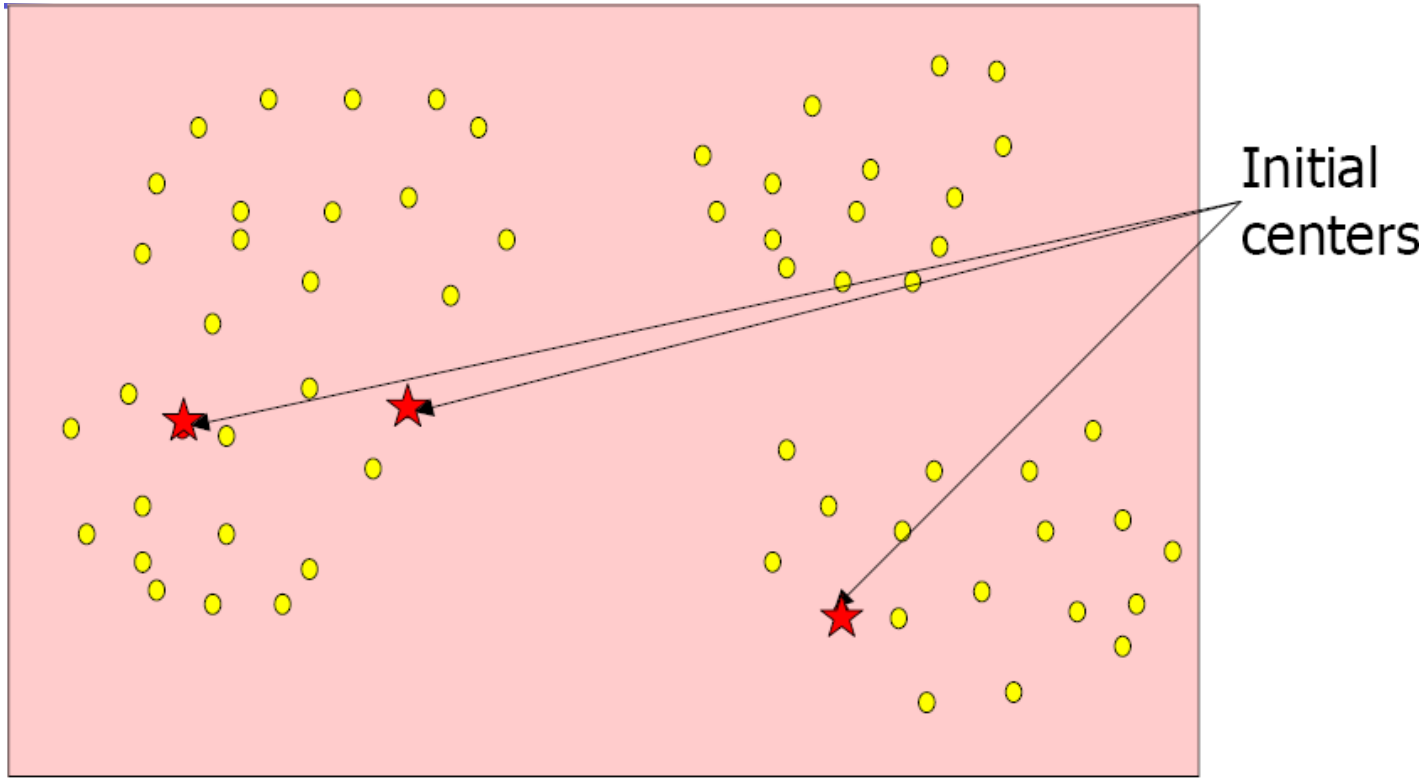
- Given k, the k-means algorithm is implemented in 4 steps:
  - Choose k random objects as initial cluster centers.
  - Assign each object to the cluster with the nearest center.
  - Recompute the center (mean point) of each cluster.
  - Go back to Step 2, stop when no more change in the cluster centers.

  K-means minimizes the mean squared error:

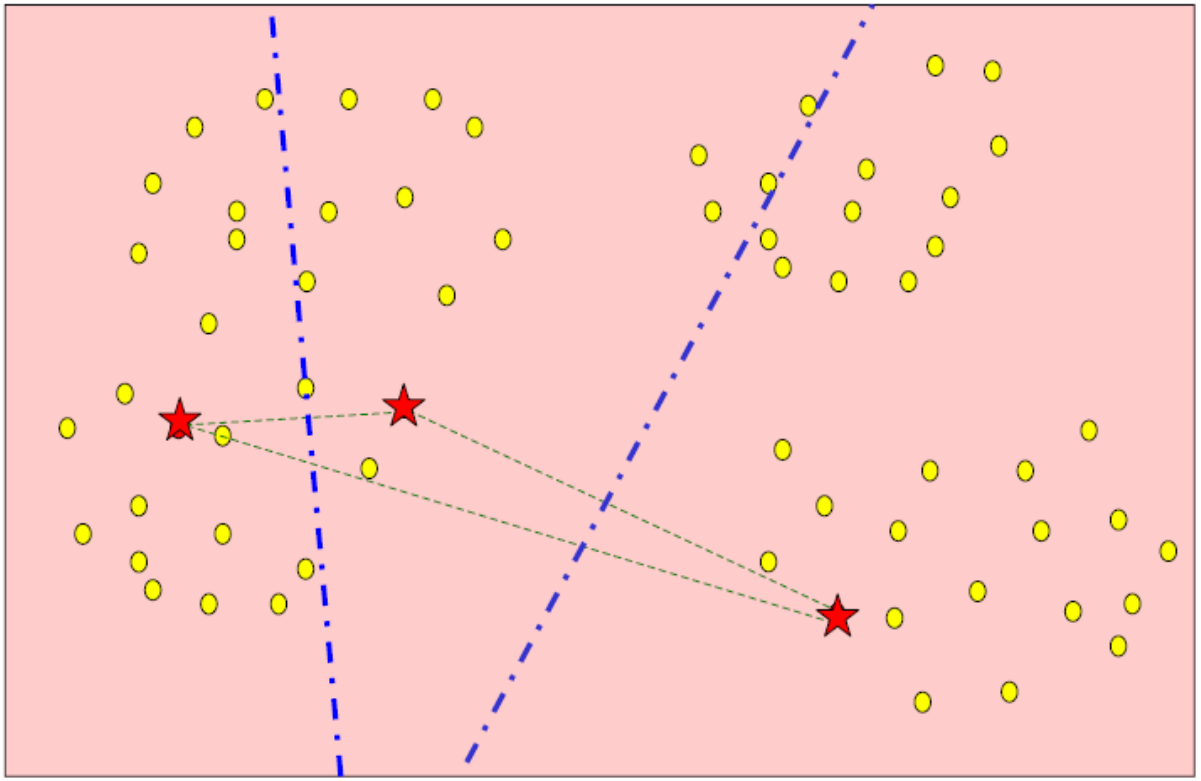
$$J = \sum_{i=1}^{k} \sum_{x_j \in C_i} D^2(v_i, x_j)$$

  - where D is a distance function like Euclidean distance, v_i denotes the center of the i th cluster.

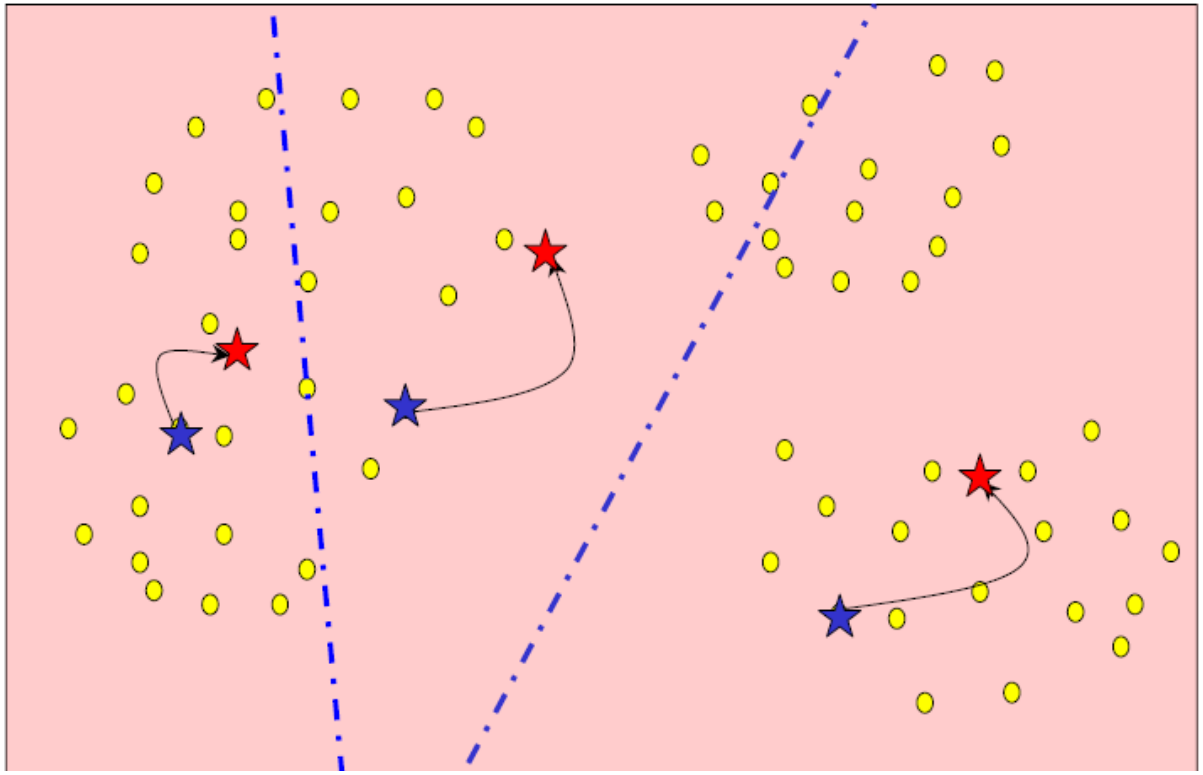
K-means - Example Initialization



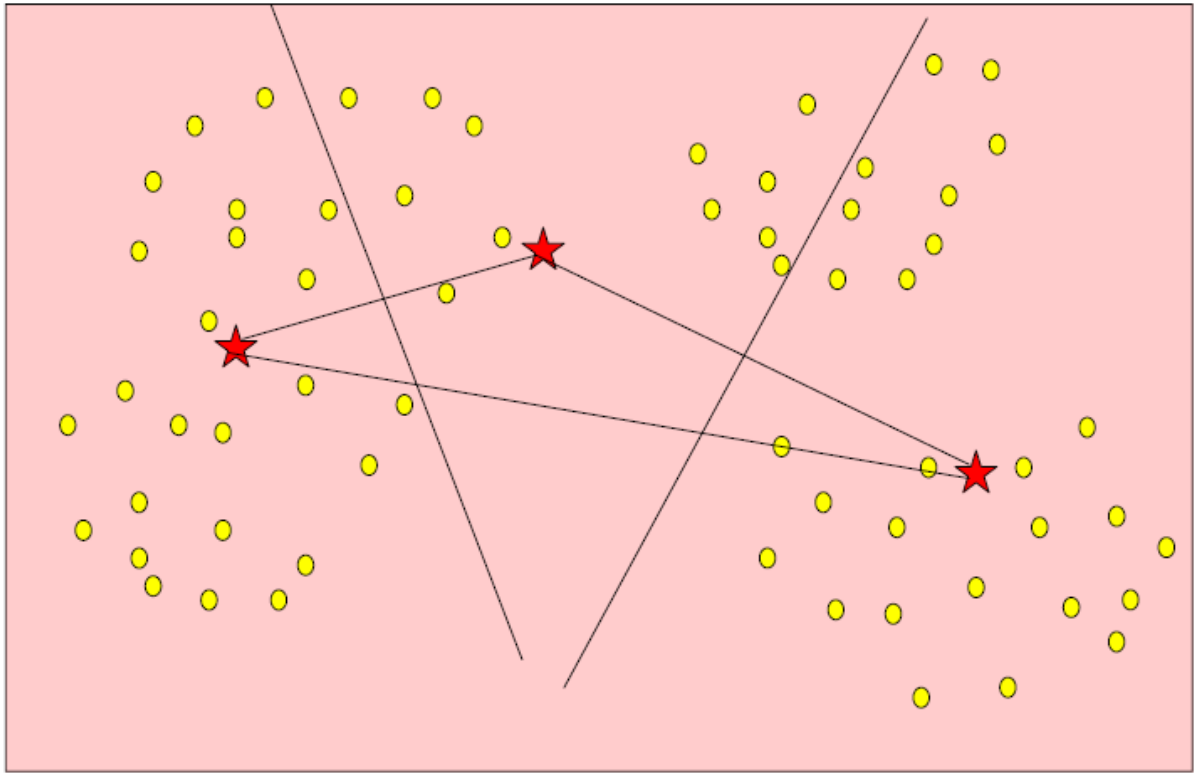
K-means - Example (Contd.)
Assign points



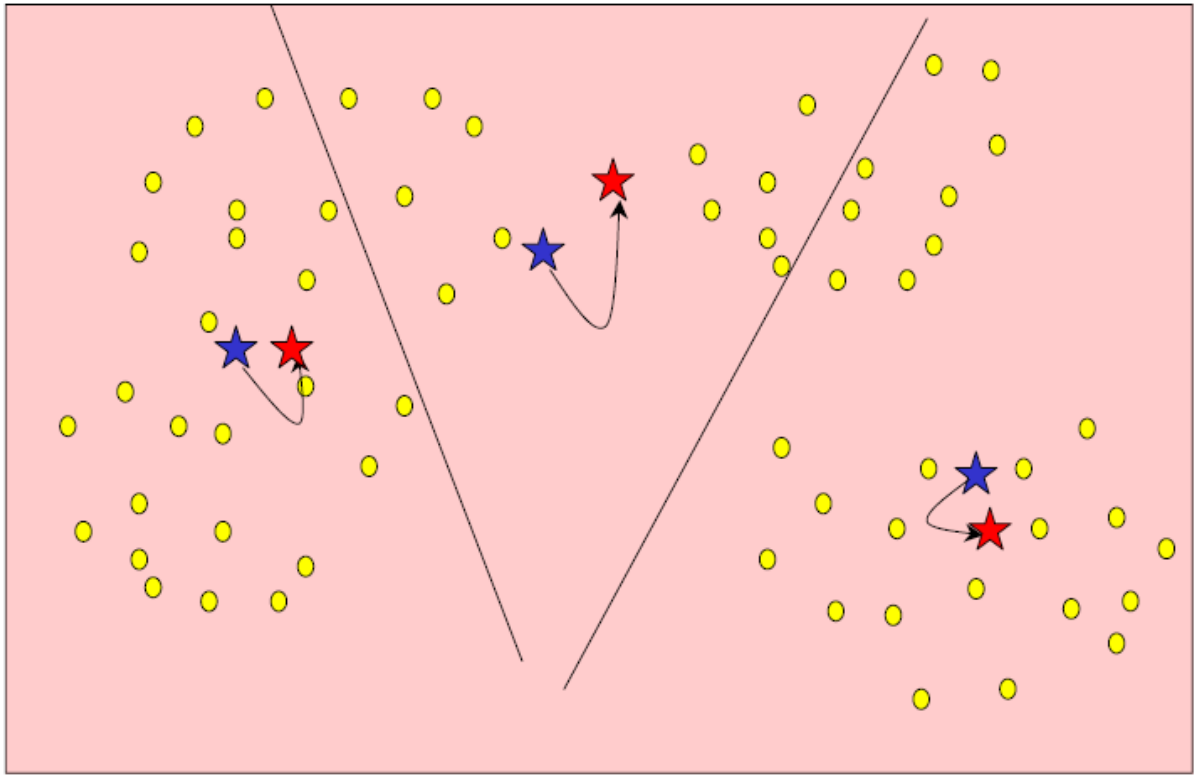
K-means - Example (Contd.)
Update centers



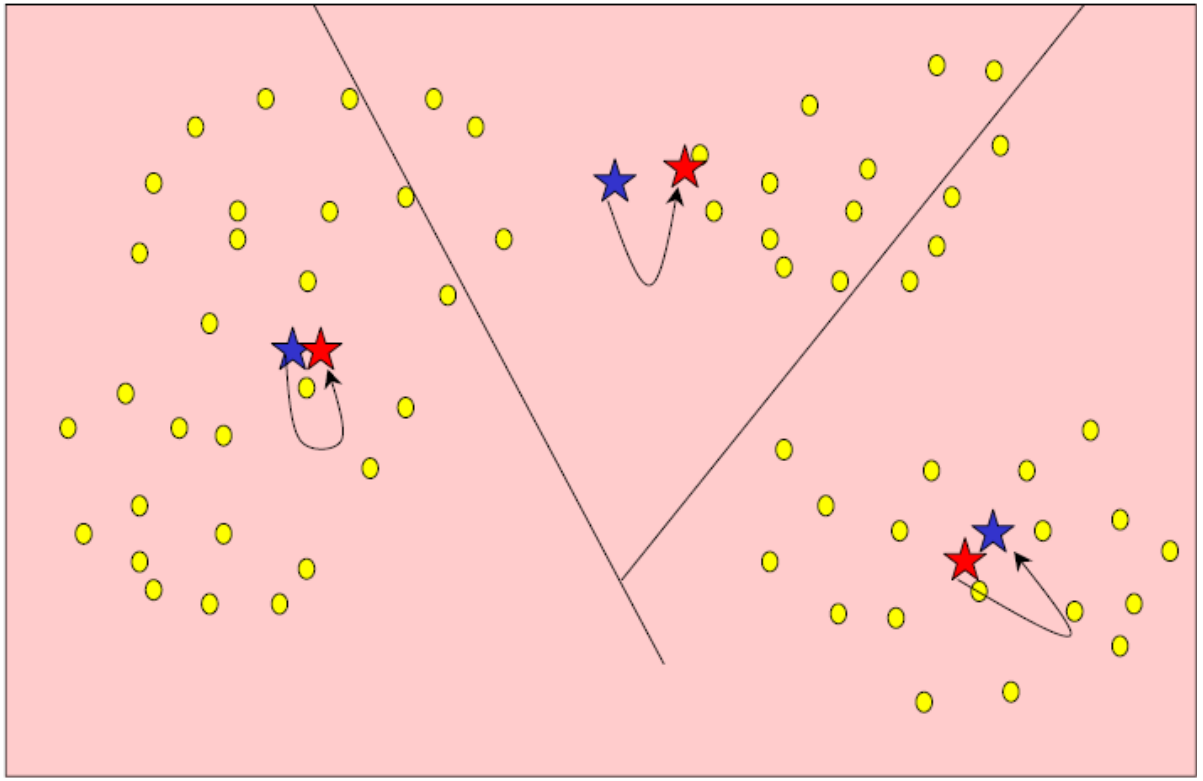
K-means - Example (Contd.)
Reassign points



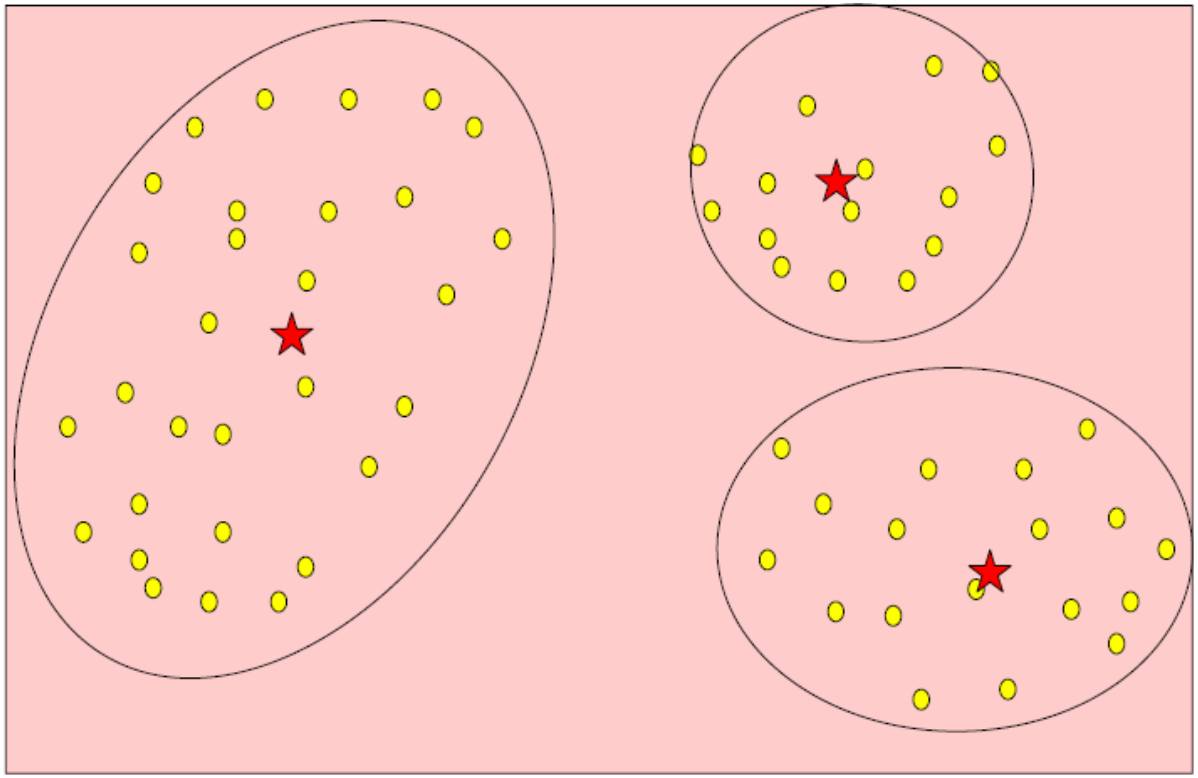
K-means - Example (Contd.)
Update centers



K-means - Example (Contd.)
Reassign points and update centers



K-means - Example (Contd.)
Final clustering after few iterations



Limitation of K-means

- Gets stuck at local optima depending on the choice of the initial cluster centers.

- Application of better optimization methods such as GA

GA based Clustering

- Representation:
  - Cluster centers encoded in the chromosomes

    For a d-dimensional space, length of chromosome = d * k

$$\{\underbrace{(v_{11}, v_{12}, \ldots, v_{1d})}_{\text{Center 1}}\ \underbrace{(v_{21}, v_{22}, \ldots, v_{2d})}_{\text{Center 2}} \ldots \underbrace{(v_{k1}, v_{k2}, \ldots, v_{kd})}_{\text{Center k}}\}$$

- Example 1:
  - Let d=2and K=3,

  i.e., two-dimensional space, number of clusters = 3
  Chromosome: {51.6 72.3 18.3 15.7 29.1 32.2} represents 3 cluster centers (51.6, 72.3), (18.3, 15.7) and (29.1, 32.2).

Population Initialization

- Initial cluster centers= k randomly selected points from the data

For each chromosome *i* in the population

    For each cluster *j*

        p=randomly chosen point from the data set;

        Population[*i*][*j*] = p;

    End

End

Fitness Computation

This consists of three phases.

- Phase 1: Cluster assignment
  - Each point is assigned to the nearest cluster center.
  - All ties are resolved arbitrarily.
- Phase 2: The cluster centers encoded in the chromosome are replaced by the mean points of the respective clusters.
- Phase 3: fitness computation
  - Compute $J = \sum_{i=1}^{k} \sum_{x_j \in C_i} D^2(v_i, x_j)$
  - Fitness = $1/J$.
  - Maximization of fitness leads to minimization of J

Fitness Computation - Example

Example 2:

- Chromosome: **{(51.6 72.3)** (18.3 15.7) (29.1 32.2)}
- The first cluster center is (51.6, 72.3).
- Let points (50.0, 70.0) and (52.0, 74.0) be also included in the first cluster besides itself i.e., (51.6, 72.3)
- Hence the newly computed cluster center becomes

  ((50.0+52.0+51.6)/3, (70.0+74.0+72.3)/3) = (51.2, 72.1).
- New cluster center replaces the previous value in chromosome, i.e., (51.2, 72.1) replaces (51.6, 72.3). Similarly other centers are updated.
- Compute mean squared error *J*.
- Fitness of chromosome = 1/*J*

Genetic Operators

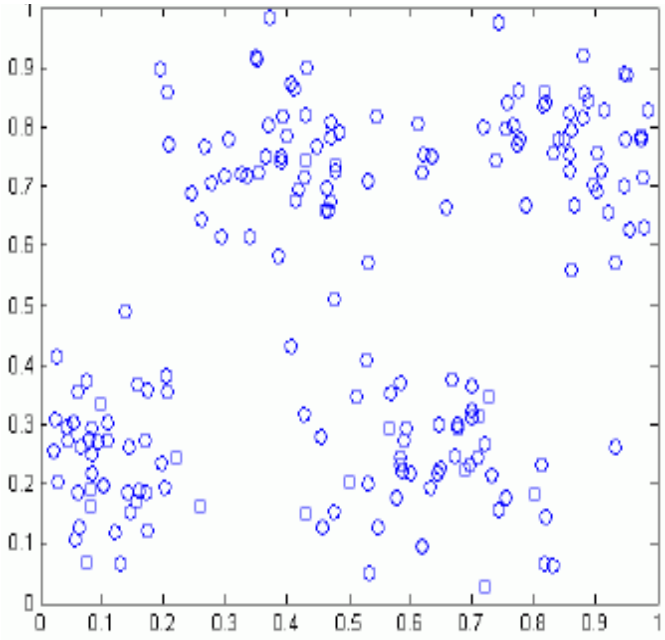
- **Selection -** Roulette wheel selection.
- **Crossover -** Single point crossover with a fixed crossover probability.
  - For chromosomes of length *k,* a random integer p is generated in the range [1, k]. The portions of the chromosomes lying to the right of p are exchanged to produce two offspring.
  - Centers are considered indivisible.
- **Mutation**
  - Since we are considering floating point representation, we use the following mutation. A number *d* in the range [0, 1] is generated with uniform distribution. If the value at a gene position is v, after mutation it becomes $v=v \pm 2*d*v$, if $v \neq 0$, $v=v \pm 2*d$, if $v=0$.

Termination

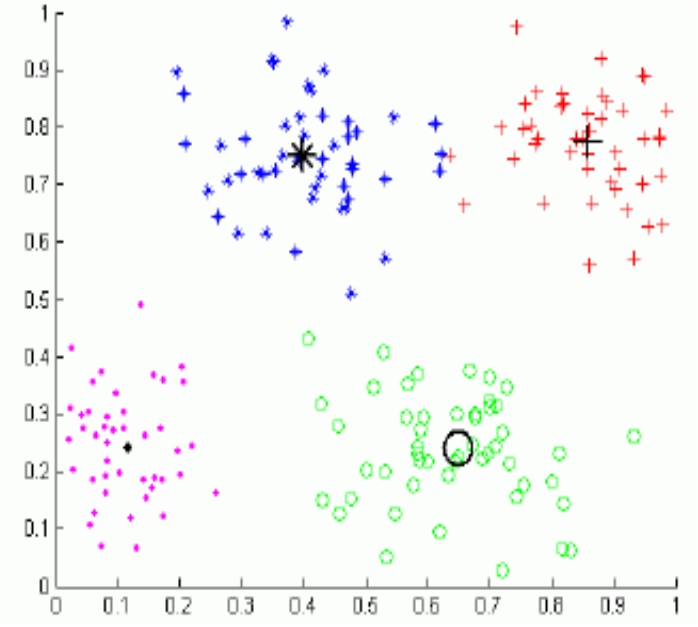
- GA clustering is run for a fixed number of generations
- Elitism incorporated
- Best string (one with the lowest J) is taken as the solution of the clustering problem.

Clustering Result - Example

N = 188, d = 2, k = 4



Input Data Set



Clustered Data

Thank You

www.anirbanm.in

anirban@klyuniv.ac.in