IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

NPTEL

Introduction to Soft Computing

GA Operators : Crossover II

Prof. Debasis Samanta
Department of Computer Science & Engineering
IIT Kharagpur

Important GA Operations

- ✓ Encoding
- ✓ Fitness evaluation and Selection
- ✓ Mating pool
- ✓ Crossover
- Mutation
- Inversion
- Convergence test

Crossover Techniques in Real Coded GA

Crossover techniques in Real coded GA

Following are the few well known crossover techniques for the real-coded GAs.

- Linear crossover
- Blend crossover
- Binary simulated crossover

Linear Crossover in Real Coded GA

Linear crossover in Real-coded GAs

- This scheme uses some linear functions of the parent chromosomes to produce the new children.

Example:

Suppose P_1 and P_2 are the two parameter's values in two parents, then the corresponding offspring values in chromosomes can be obtained as

$$C_i = \alpha_i P_1 + \beta_i P_2$$

where $i = 1, 2 \cdots n$ (number of children).
α_i and β_i are some constants decided by the user.

Linear crossover: An example

Example :

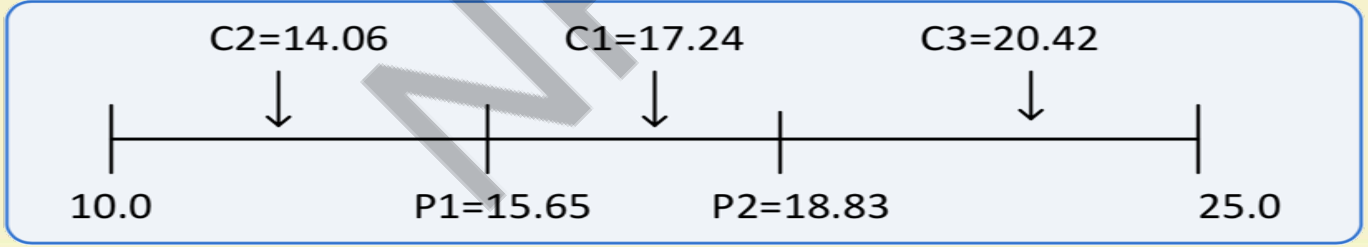
Suppose $P_1 = 15.65$ and $P_2 = 18.83$ and $\alpha_1 = \beta_1 = 0.5$

$\alpha_2 = 1.5$ and $\beta_2 = -0.5$

$\alpha_3 = -0.5$ and $\beta_3 = 1.5$

Answer :

$$C_1 = 0.5 \times (P_1 + P_2) = 17.24$$
$$C_2 = 1.5 \times P_1 - 0.5 \times P_2 = 14.06$$
$$C_3 = -0.5 \times P_1 + 1.5 \times P_2 = 20.24$$



Advantages and limitations

Advantages:

- It is simple to calculate and hence faster in computation.
- Can allow to generate a large set of offspring from two parent values.
- Controls are possible to choose a wide-range of variations.

Limitations:

- Needs to be decided the values of α_i and β_i
- It is difficult for the inexperienced users to decide the right values for α_i and β_i
- If α_i and β_i values are not chosen properly, the solution may stuck into a local optima.

Blend Crossover in Real Coded GA

Blend crossover in Real-coded GAs

This scheme can be stated as follows.

1) Let P_1 and P_2 are the two parameter's values in two parent's chromosomes, such that $P_1 < P_2$.

2) Then the blend crossover scheme creates the children solution lying in the range

$$\langle \{P_1 - \alpha(P_2 - P_1)\} \cdots \{P_2 - \alpha(P_2 - P_1)\} \rangle$$

where α is a constant to be decided so that children solution do not come out of the range of domain of the said parameter.

Blend crossover in Real-coded GAs

3) Another parameter γ has to be identified by utilizing the α and a random number r in the range of $(0.0, 1.0)$ both exclusive like the following:

$$\gamma = (1 + 2\alpha) r - \alpha$$

4) The children solutions C_1 and C_2 are determined from the parents as follows,

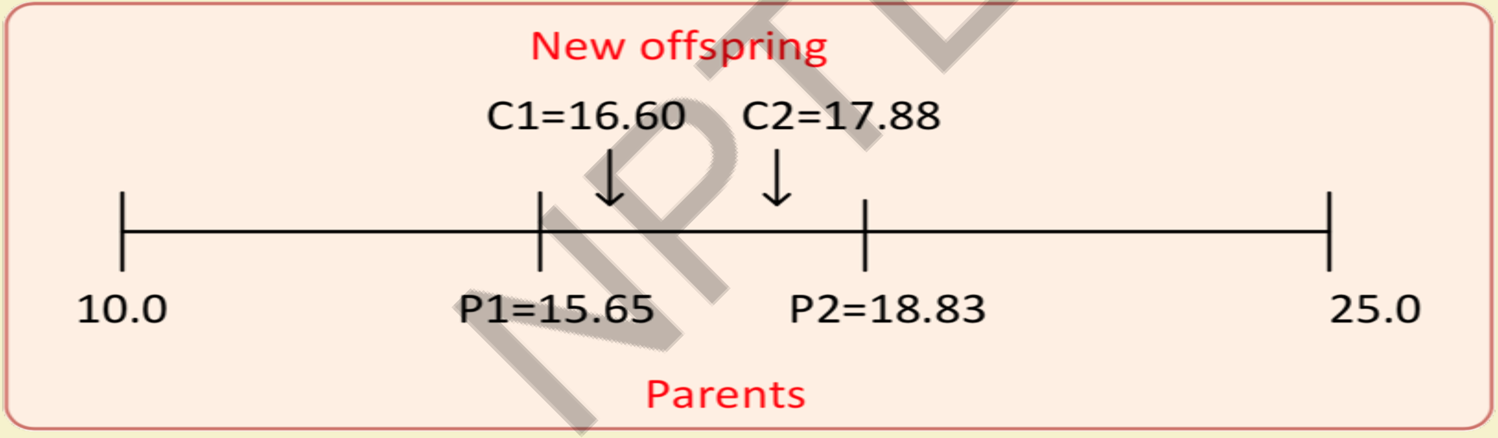
$$C_1 = (1 - \gamma) P_1 + \gamma P_2$$
$$C_2 = (1 - \gamma) P_2 + \gamma P_1$$

Blend crossover : An example

Example :

$P_1 = 15.65$ and $P_2 = 18.83$

$\alpha = 0.5$ and $\gamma = 0.6$



Advantages and limitations

Advantages:

- It is simple to calculate and also faster in computation.
- Can allow to generate a large set of offspring from two parent values.
- Controls are possible to choose a wide-range of variations.

Limitations:

- Needs to be decided the values of α and γ
- It is difficult for the inexperienced users to decide the right values for α and γ
- If α and γ values are not chosen properly, the solution may stuck into a local optima.

Binary Simulated Crossover in Real GA

Simulated binary crossover in Real-coded GAs

- This scheme is based on the probability distribution of generated children solution from the given parents.
- A spread factor α is used to represent the spread of the children solutions with respect to that of the parents, as given below.

$$\alpha = \frac{C_1 - C_2}{P_1 - P_2}$$

Here, P_1 and P_2 are represent the parent points and C_1 and C_2 are two children solutions.

Simulated binary crossover in Real-coded GAs

Three different cases may occurs:

- Case 1: $\alpha < 1$ (Contracting Crossover)
  The spread of children is less than the parents.
- Case 2: $\alpha > 1$ (Expanding Crossover)
  The spread of children is more than the parents.
- Case 3: $\alpha = 1$ (Stationary Crossover)
  The spread of children is same as that of the parents.

Simulated Binary Crossover

Probability Distribution:

- **Case 1:** For Contracting Crossover

$$C(\alpha) = 0.5(q + 1)\alpha^2$$

- **Case 2:** For Expanding Crossover

$$E(\alpha) = 0.5(q + 1)\frac{1}{\alpha^{q+2}}$$

Here, q is a parameter chosen by the user.

Simulated binary crossover in Real-coded GAs

Following steps are used to create two children solutions C_1 and C_2 from the parents P_1 and P_2.

1. Create a random number $r \in \{0.0 \cdots 1.0\}$
2. Determine α' such that

$$\int_0^{\alpha'} C(\alpha)d\alpha = r, \quad if\ r < 0.5 \qquad \text{and}$$

$$\int_1^{\alpha'} E(\alpha)d\alpha = r, \qquad if\ r > 0.5$$

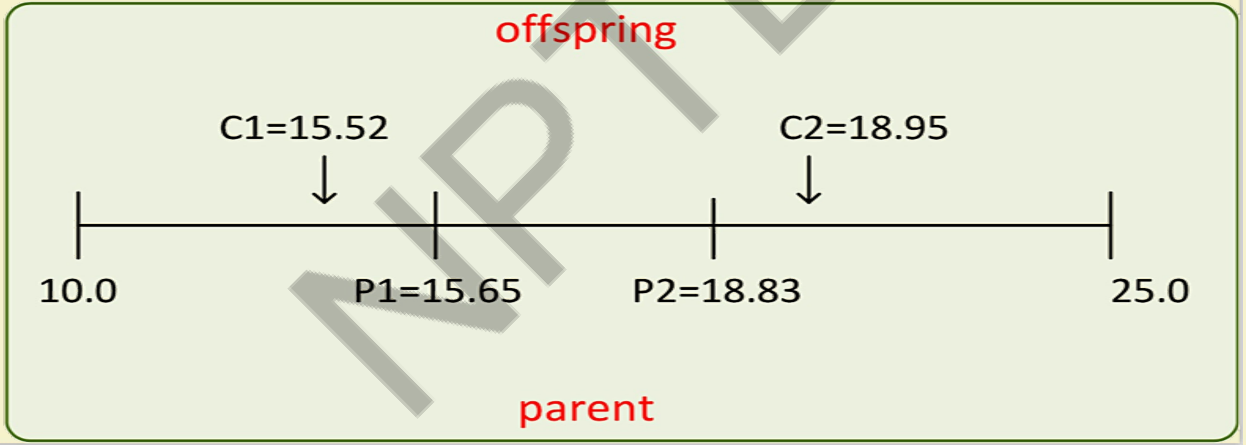
3. Using the value of α' obtain two children solution as follows
   - $C_1 = 0.5[(P_1 + P_2) - \alpha'|P_2 - P_1|]$
   - $C_2 = 0.5[(P_1 + P_2) + \alpha'|P_2 - P_1|]$

Simulated binary crossover in Real-coded GAs

Example:

$$P_1 = 15.65 \qquad P_2 = 18.83$$
$$q = 2 \qquad \alpha' = 1.0772$$

Assuming expanding crossover with $r > 0.5$



Advantages

Advantages

- We can generate a large number of offspring from two parents.
- More explorations with diverse offspring.
- Results are accurate and usually terminated with global optima.
- Termination with a less number of iterations.
- Crossover techniques are independent of the length of the chromosome.

Advantages and limitations

Limitations

- Computationally expensive compared to binary crossover.
- If proper values of parameters involved in the crossover techniques are not chosen judiciously, then it may lead to premature convergence with not necessarily optimum solutions.

Thank You!!

IIT KHARAGPUR

NPTEL ONLINE CERTIFICATION COURSES

NPTEL

Introduction to Soft Computing

GA Operators : Crossover III

Prof. Debasis Samanta
Department of Computer Science & Engineering
IIT Kharagpur

Important GA Operations

- ✓ Encoding
- ✓ Fitness evaluation and Selection
- ✓ Mating pool
- ✓ Crossover
- Mutation
- Inversion
- Convergence test

Crossover Techniques in Order GAs

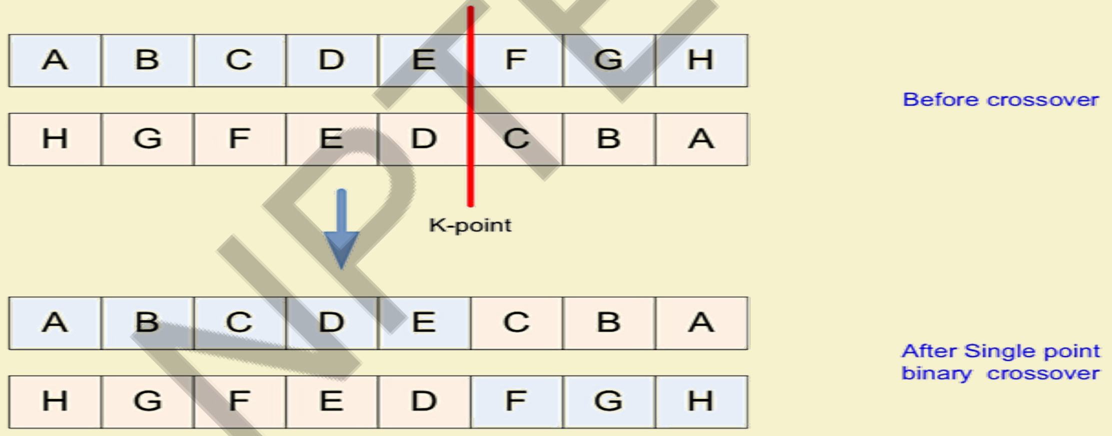
Crossover techniques in order GA

- Any binary crossover techniques are not applicable to Order coded GAs.

Example:

Reference: TSP

Consider any two chromosomes with Order-coded encoding scheme



Here, the offspring are not valid chromosomes

Crossover techniques in order GA

- Since, sequence of gene values are important, Real-coded crossover techniques, which are to produce real number from two given real numbers are also not applicable to Order-coded GAs.

Crossover techniques in order GA

Some important crossover techniques in Order-coded GAs are:

- Single-point order crossover
- Two-point order crossover
- Precedence-preservation crossover (PPX)
- Position based crossover
- Edge recombination crossover

Crossover techniques in order GA

Assumptions: For all crossover techniques, we assume the following:

- Let L be the length of the chromosome.
- P_1 and P_2 are two parents (selected from the mating pool).
- C_1 and C_2 denote offspring (initially empty).

Single Point Crossover in Order GAs

Single point order crossover

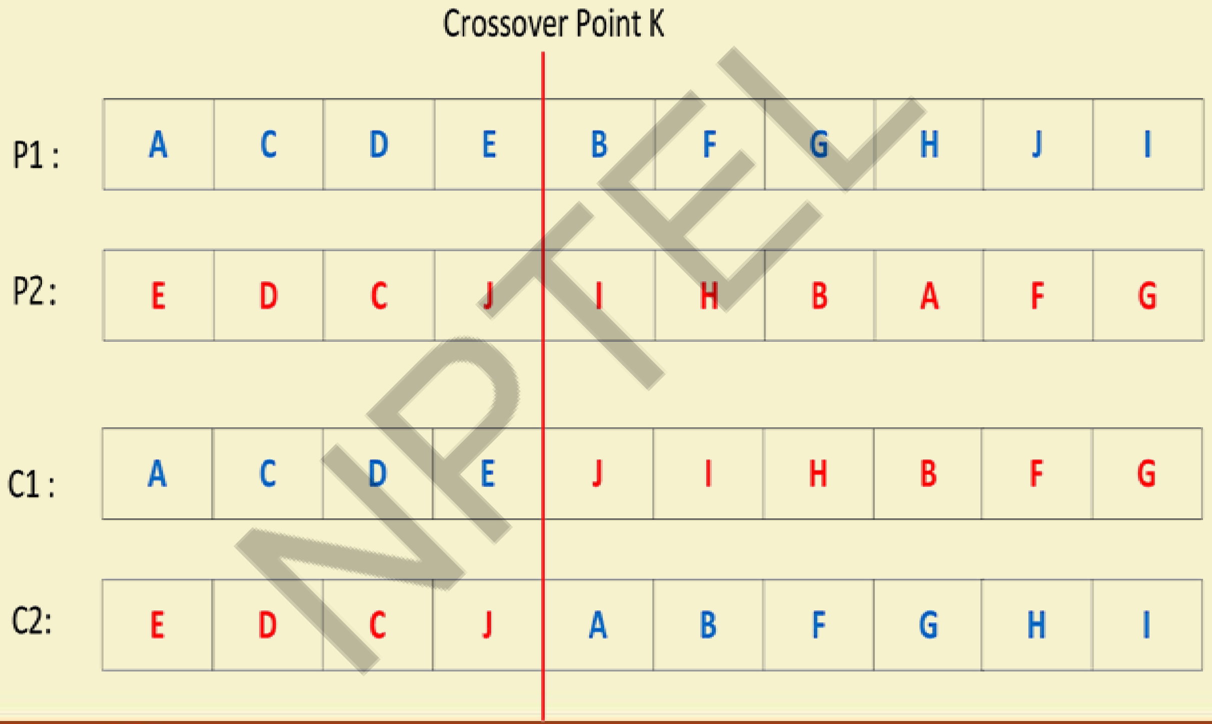
Given two parents P_1 and P_2 with chromosome length, say L.

Steps:

1) Randomly generate a crossover point K such that $(1 < K < L)$.
2) Copy the left schema of P_1 into C_1 (initially empty) and left schema of P_2 into C_2 (also initially empty).
3) For the schema in the right side of C_1, copy the gene value from P_2 in the same order as they appear but not already present in the left schema.
4) Repeat the same procedure to complete C_2 from P_1.

Single point order crossover: Illustration

Example :



P1 :	A	C	D	E	B	F	G	H	J	I
P2:	E	D	C	J	I	H	B	A	F	G
C1 :	A	C	D	E	J	I	H	B	F	G
C2:	E	D	C	J	A	B	F	G	H	I

Two Point Crossover in Order GAs

Two-point order crossover

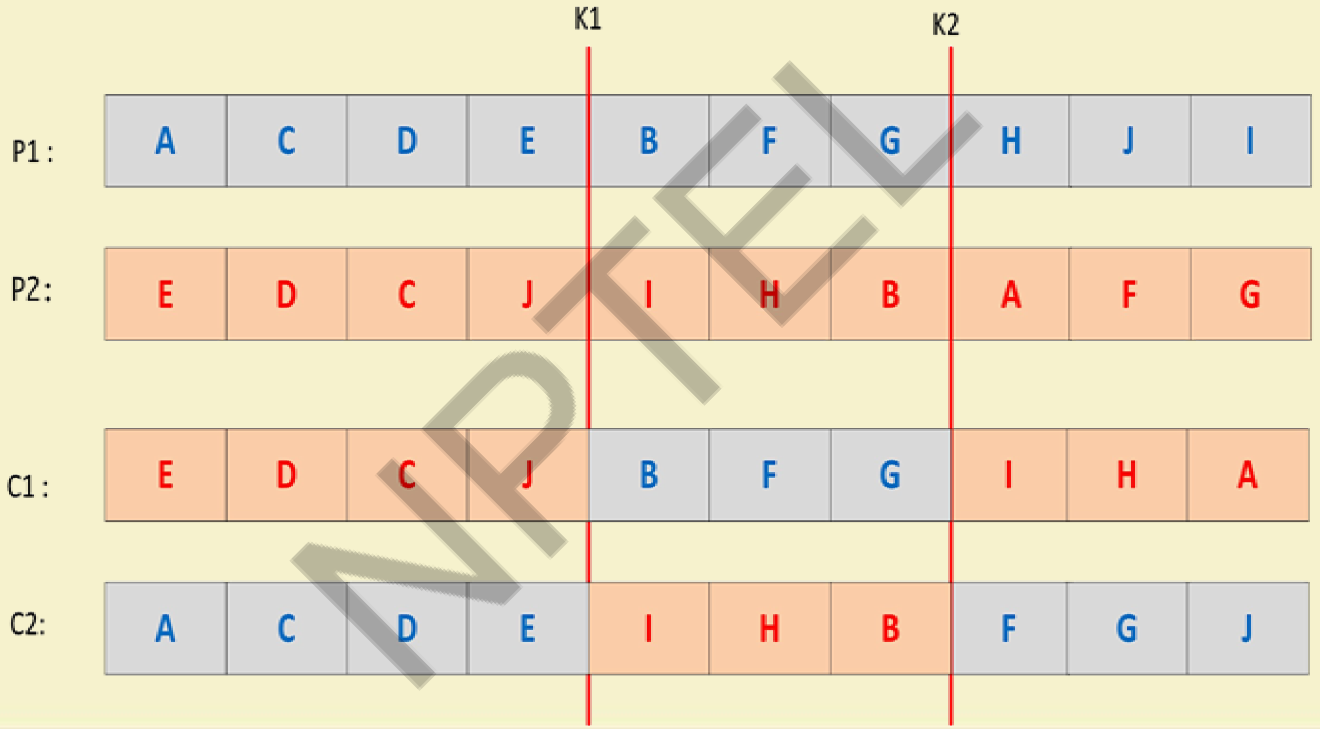
It is similar to the single-point order crossover, but with two k–points.

Steps:

1) Randomly generate two crossover points K_1 and K_2. $1 < K_1, K_2 < L$
2) The schema in middle of P_1 and P_2 are copied into C_1 and C_2 (initially both are empty), respectively in the same location as well as in the same order.
3) The remaining schema in C_1 and C_2 are copied from P_2 and P_1, respectively, so that an element already selected in child solution does not appear again.

Two-point order crossover: Illustration

Example :



PPX Crossover in Order GAs

Precedence preservation order crossover

Let the parent chromosomes be P_1 and P_2 and the length of chromosomes be L.

Steps:

a) Create a vector V of length L randomly filled with elements from the set $\{1, 2\}$.

b) This vector defines the order in which genes are successfully drawn from P_1 and P_2 as follows.

  1) We scan the vector V from left to right.

  2) Let the current position in the vector V be i (where $i = 1,2,\cdots,L$).

  3) Let j (where $j = 1,2,\cdots,L$) and k (where $j = 1,2,\cdots,L$) denotes the j^{th} and k^{th} gene of P_1 and P_2, respectively. Initially $j = k = 1$.

Precedence preservation order crossover

4) If i^{th} value is 1 then

   Delete j^{th} gene value from P_1 and as well as from P_2 and append it to the offspring (which is initially empty).

5) Else

   Delete k^{th} gene value from P_2 and as well as from P_1 and append it to the offspring.

6) Repeat Step 2 until both P_1 and P_2 are empty and the offspring contains all gene values.

Precedence preservation order crossover: Example

Random Vector σ	2	1	1	2	1	1	2	2	1	2
P1 :	A	C	D	E	B	F	G	H	J	I
P2 :	E	D	C	J	I	H	B	A	F	G
C1 :	E	C	D	J	B	F	H	A	I	G
C2:	?									

Note : We can create another offspring following the alternative rule for 1 and 2.

Position-Based Crossover in Order GAs

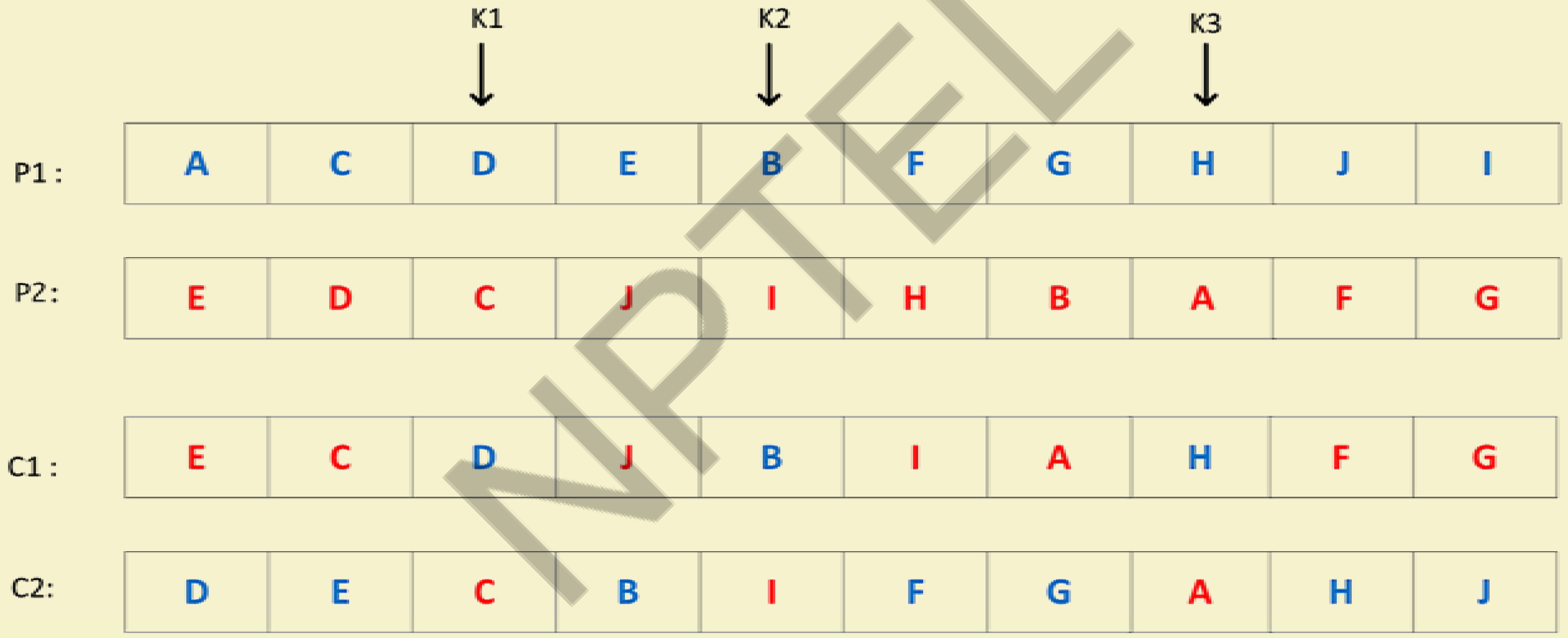
Position-based order crossover

Steps:

1) Choose n crossover points $K_1, K_2 \cdots K_n$ such that $n \ll L$, the length of chromosome.
2) The gene values at $K_1^{th}, K_2^{th}, \ldots K_n^{th}$ positions in P_1 are directly copied into offspring C_1 (Keeping their position information intact).
3) The remaining gene values in C_1 will be obtained from P_2 in the same order as they appear there except they are already not copied from P_1.
4) We can reverse the role of P_1 and P_2 to get another offspring C_2.

Position-based order crossover : Example

Let us consider three k −points namely K_1, K_2 and K_3 in this example.



P1 :	A	C	D	E	B	F	G	H	J	I
P2 :	E	D	C	J	I	H	B	A	F	G
C1 :	E	C	D	J	B	I	A	H	F	G
C2:	D	E	C	B	I	F	G	A	H	J

Edge Recombination Order Crossover

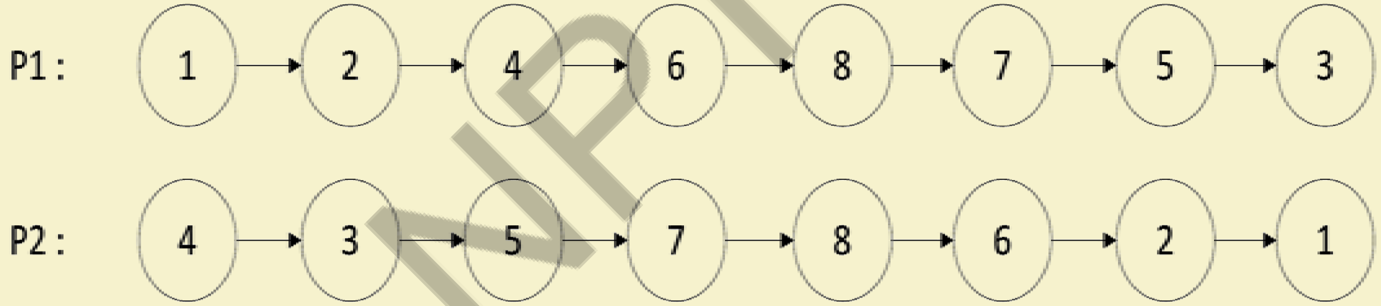
Edge recombination order crossover

- This crossover technique is used to solve TSP problem when the cities are not completely connected to each other.
- In this technique, an edge table which contains the adjacency information (but not the order).
- In other words, edge table provides connectivity information.

Edge recombination order crossover: Illustration

Example

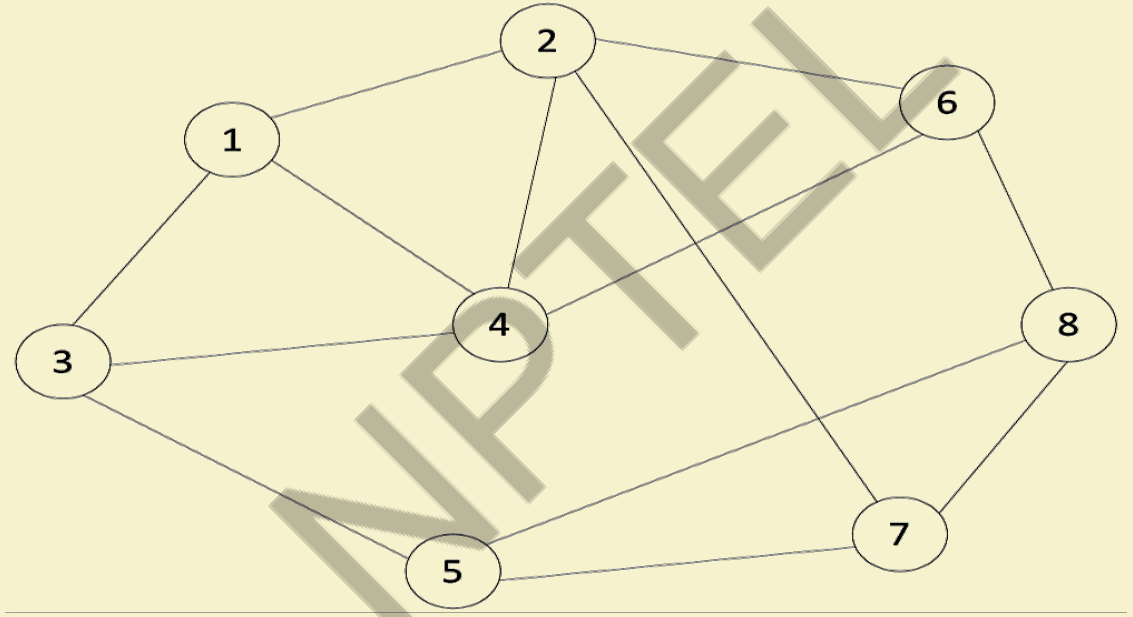
- Let us consider a problem instance of a TSP with 8 cities.
- Assume any two chromosome P_1 and P_2 for the mating.



$$\begin{tikzcd}
P1: & 1 \arrow[r] & 2 \arrow[r] & 4 \arrow[r] & 6 \arrow[r] & 8 \arrow[r] & 7 \arrow[r] & 5 \arrow[r] & 3 \\
P2: & 4 \arrow[r] & 3 \arrow[r] & 5 \arrow[r] & 7 \arrow[r] & 8 \arrow[r] & 6 \arrow[r] & 2 \arrow[r] & 1
\end{tikzcd}$$

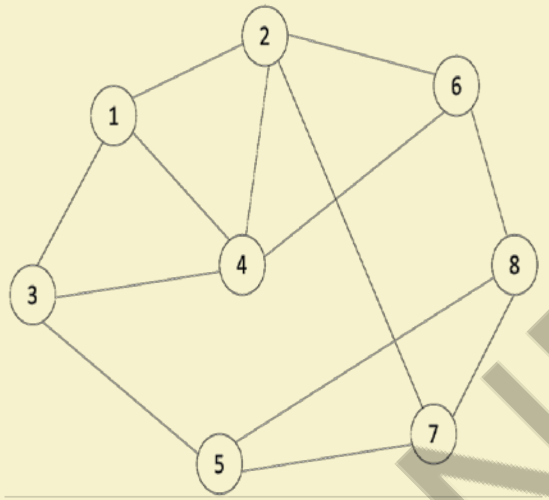
Edge recombination order crossover: Illustration

Connectivity graph:



Edge recombination order crossover: Illustration

Edge table for the connectivity graph:



City	Connectivity
1	2 4 3
2	1 4 7 6
3	1 4 5
4	1 2 3 6
5	3 7 8
6	2 4 8
7	2 5 8
8	5 6 7

Edge recombination order crossover: Illustration

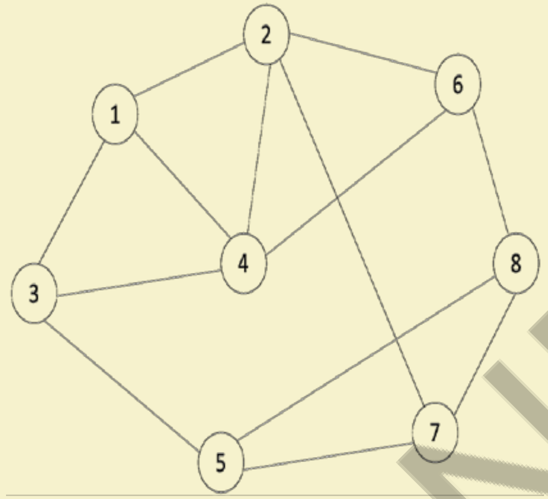
Steps:

Let the child chromosome be C_1 (initially empty).

1) Start the child tour with the starting city of P_1. Let this city be X.
2) Append city X to C .
3) Delete all occurrences of X from the connectivity list of all cities (right-hand column).
4) From city X choose the next city say Y, which is in the list of minimum (or any one, if there is no choice) connectivity links.
5) Make $X = Y$ [i.e., new city Y becomes city X].
6) Repeat Steps 2-5 until the tour is complete.
7) End

Edge recombination order crossover: Illustration

Edge table for the connectivity graph:



City	Connectivity
1	2 4 3
2	1 4 7 6
3	1 4 5
4	1 2 3 6
5	3 7 8
6	2 4 8
7	2 5 8
8	5 6 7

Thank You!!

IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

NPTEL

Introduction to Soft Computing

Mutation and Fitness Scaling in GAs

Debasis Samanta
Department of Computer Science and Engineering
IIT KHARAGPUR

Important GA Operations

- ✓ Encoding
- ✓ Fitness Evaluation and Selection
- ✓ Mating pool

Reproduction

- ✓ Crossover
- • Mutation
- • Inversion

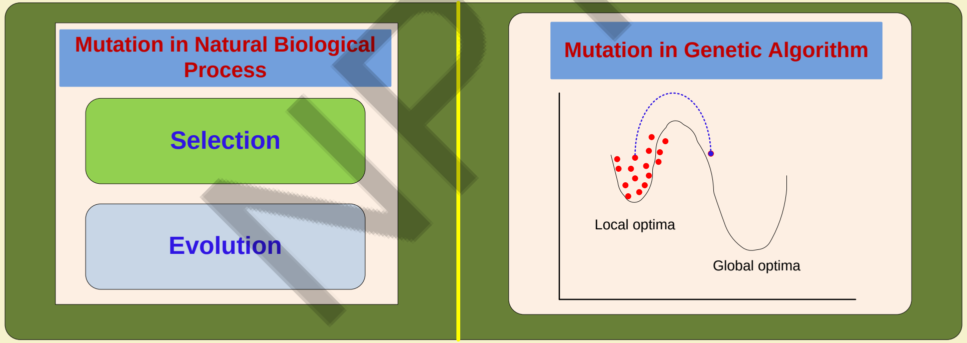
Convergence test

This lecture includes …

1) Encoding
2) Fitness evaluation and Selection
3) Mating pool
4) Crossover
5) Mutation
6) Inversion
7) Convergence test
8) Fitness scaling

Mutation Operation

1) In genetic algorithm, the mutation is a genetic operator used to maintain genetic diversity from one generation of a population (of chromosomes) to the next.
2) It is analogues to biological mutation.
3) In GA, the concept of biological mutation is modelled artificially to bring a local change over the current solutions.



Mutation in Natural Biological Process
Selection
Evolution
Mutation in Genetic Algorithm
Local optima
Global optima

Mutation Operation in GAs

Like different crossover techniques in different GAs there are many variations in mutation operations.

1) **Binary Coded GA :**
    - Flipping
    - Interchanging
    - Reversing
2) **Real Coded GA :**
    - Random mutation
    - Polynomial mutation
3) **Order GA :**
4) **Tree-encoded GA :**

Mutation Operation in Binary coded GA

1) In binary-coded GA, the mutation operator is simple and straight forward.
2) In this case, one (or a few) $1(s)$ is(are) to be converted to $0(s)$ and vice-versa.
3) A common method of implementing the mutation operator involves generating a random variable called mutation probability (μ_ρ) for each bit in a sequence.
4) This mutation probability tells us whether or not a particular bit will be mutated (i.e., modified).

Note:

1) To avoid a large deflection, μ_ρ is generally kept to a low value.
2) It is varied generally in the range of $\frac{0.1}{L}$ to $\frac{1.0}{L}$, where L is the string length.

Mutation in Binary-coded GA : Flipping

- Here, a mutation chromosome of the same length as the individual's chromosome is created with a probability μ_p in the bit.
- For a bit in a mutation chromosome, the corresponding bit in the current chromosome is flipped (0 to 1 or 1 to 0) and mutated chromosome is produced.

1	0	1	1	0	0	1	0

offspring

1	0	0	0	1	0	0	1

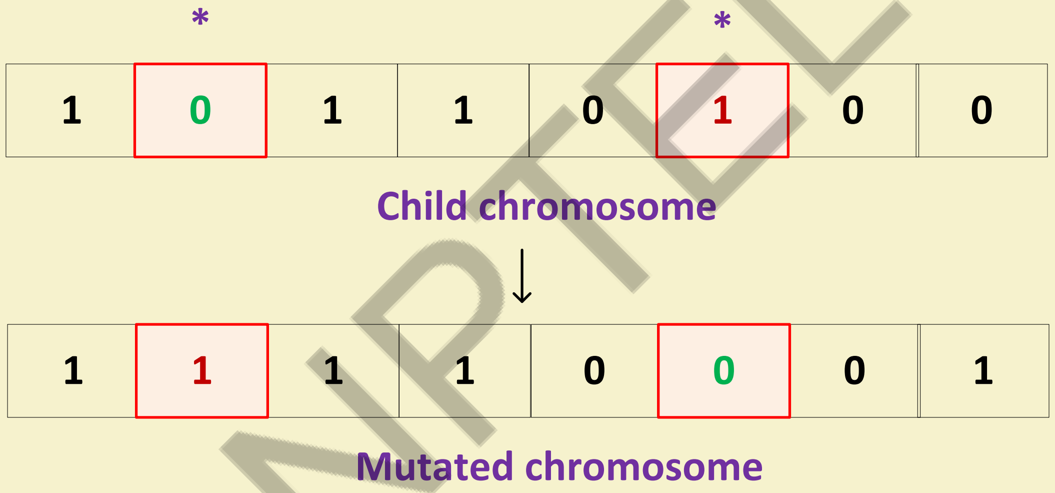
Mutation probability

0	0	1	1	1	0	1	1

Mutated offspring

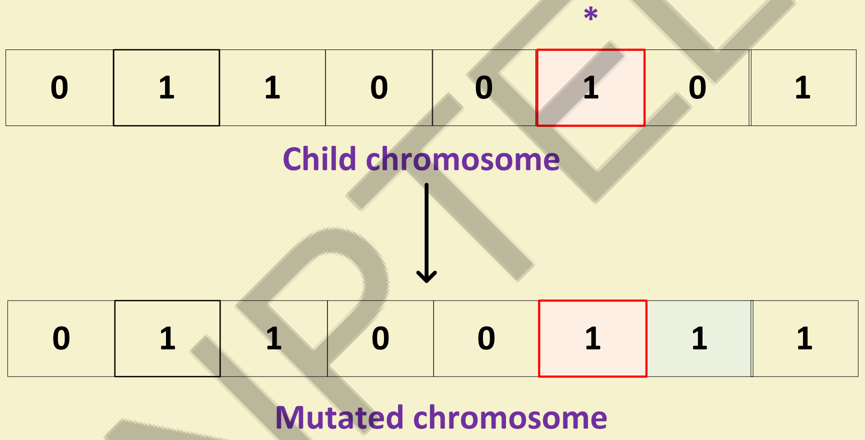
Binary-coded GA : Interchanging

- Two positions of a child's chromosome are chosen randomly and the bits corresponding to those position are interchanged.



Mutation in Binary-coded GA : Reversing

- A positions is chosen at random and the bit next to that position are reversed and mutated child is produced.



Mutation Operation in GAs

Like different crossover techniques in different Gas, there are many variations in mutation operations.

- **Binary Coded GA**
  - Flipping
  - Interchanging
  - Reversing
- **Real-coded GA**
  - Random mutation
  - Polynomial mutation
- **Order GA**
- **Tree-encoded GA**

Mutation in Real-coded GA : Random mutation

- Here, mutated solution is obtained from the original solution using the following rule.

$$P_{mutated} = P_{original} + (r - 0.5) \times \Delta$$

- Where r is a random number lying between 0.0 and 1.0 and Δ is the maximum value of the perturbation decided by the user.

Example :

$P_{original} = 15.6$

$r = 0.7$

$\Delta = 2.5$

then $P_{mutated} = 15.6 + (0.7 - 0.5) \times 2.5 = 16.1$

Mutation in Real-coded GA : Polynomial mutation

It is a mutation operation based on the polynomial distribution. Following steps are involved.

1) Calculate a random number r lying between 0.0 and 1.0
2) Calculate the perturbation factor δ using the following rule

$$\delta = \begin{cases} (2r)^{\frac{1}{q+1}} - 1 & if\ r < 0.5 \\ 1 - [2(1-r)]^{\frac{1}{q+1}} & if\ r \geq 0.5 \end{cases}$$

where q is a exponent (positive or negative value) decided by the user.

3) The mutated solution is then determined from the original solution as follows

$$P_{mutated} = P_{original} + \delta \times \Delta$$

Where Δ is the user defined maximum perturbation allowed between the original and mutated value.

Mutation in Real-coded GA : Polynomial mutation

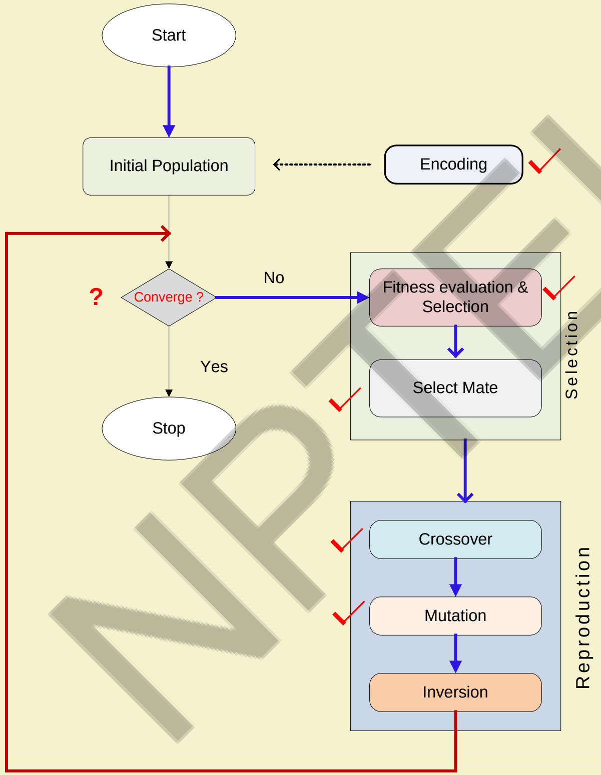
Example :

$P_{original} = 15.6\ r = 0.7\ q = 2, \Delta = 1.2$ then $P_{mutated} = ?$

$$\delta = 1 - [2(1-r)]^{\frac{1}{q+1}} = 0.1565$$

$$P_{mutated} = 15.6 \times 0.1565 \times 1.2 = 15.7878$$

Revisiting the flow in GA



Termination and/or convergence criteria

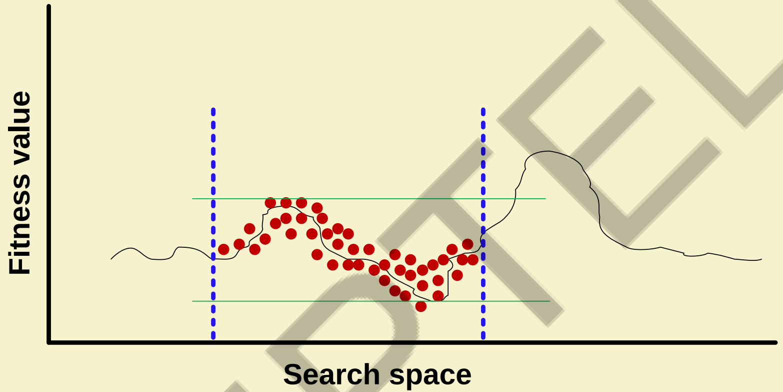
Each iteration should be tested with some convergence test. Commonly known convergence test or termination conditions are :

1) A solution is found that satisfies the objective criteria.
2) Fixed number of generation is executed.
3) Allocated budget (such as computation time) reached.
4) The highest ranking solution fitness is reaching or has reached a plateau such that successive iterations no longer produce better result.
5) Manual inspection.
6) Combination of the above.

Fitness Scaling in GA

Issue with fitness values

Now, let us look into another scenario, which is depicted in the following figure.



- Here, the fitness values are with narrower range of values.
- In this case, successive iterations will not show any improvement and hence stuck at local optima/premature termination/inaccurate result.

Summary of observations

It is observed that

- If fitness values are too far apart, then it will select several copies of the good individuals and many other worst individual will not be selected at all.
- This will tend to fill the entire population with very similar chromosomes and will limit the ability of the GA to explore large amount of the search space.
- If the fitness values are too close to each other, then the GA will tend to select one copy of each individual, consequently, it will not be guided by small fitness variations and search scope will be reduced.

Why fitness scaling?

- ✓ As a way out we can think for crossover or mutation (or both) with a higher fluctuation in the values of design parameter.
  - ➢ This leads to a chaos in searching.
  - ➢ May jump from one local optima to other.
  - ➢ Needs a higher number of iterations.
- ✓ As an alternative to the above, we can think for **fitness scaling** strategy.

Fitness scaling is used to scale the raw fitness values so that the GA sees a **reasonable** amount of difference in the scaled fitness values of the best versus worst individuals.

Approaches to fitness scaling

In fact, fitness scaling is a sort of discriminating operation in GA. Few algorithms are known for fitness scaling:

- Linear scaling
- Sigma scaling
- Power law scaling

Note:

The fitness scaling is useful to avoid premature convergence, and slow finishing.

Linear scaling

Algorithm

Input : $F = \{f_1, f_2, \dots \dots f_N\}$ is a set of raw fitness values of N individuals in a population.

Output : $F' = \{f'_1, f'_2, \dots \dots f'_N\}$ is a set of fitness values after scaling

Steps :

1) Calculate the average fitness value

$$\bar{f} = \frac{\sum_{i=1}^{n} f_i}{N}$$

2) Find $f_{max} = MAX(F)$, Find the maximum value in the set F.

3) Find $f_{min} = MIN(F)$, Find the minimum value in the set F.

Linear scaling

4) Calculate the following,

$$a = \frac{\bar{f}}{f_{max} - \bar{f}},$$
$$b = \frac{\bar{f} \times f_{min}}{f_{min} - \bar{f}}$$

5) For each $f_i \in F$ do

$$f_i' = a \times f_i + b$$
$$F' = F' \cup f_i'$$

where F' is initially empty.

6) End

Linear scaling

Note :

1) For better scaling it is desirable to have $\bar{f} = \bar{f'}$

2) In order not to follow dominance by super individuals, the number of copies can be controlled with $f'_{max} = C \times \bar{f'}$ where $C = \frac{f_{max} - f_{min}}{\bar{f} - f_{min}}$

Sigma scaling

Algorithm

Input : $F = \{f_1, f_2, \dots \dots f_N\}$ is a set of raw fitness values of N individuals in a population.

Output : $F' = \{f'_1, f'_2, \dots \dots f'_N\}$ is a set of fitness values after scaling

Steps :

1) Calculate the average fitness value

$$\bar{f} = \frac{\sum_{i=1}^{n} f_i}{N}$$

2) Determine reference worst-case fitness value f_w such that

$$f_w = \bar{f} + S \times \sigma$$

where $\sigma = STD(F)$, is the standard deviation of the fitness of population and S is a user defined factor called sigma scaling factor (Usually $1 \le S \le 5$)

Sigma scaling

```
3)  Calculate $f_i'$ as follows,
        For each $f_i \in F$ do
            $f_i' = f_w - f_i, if (f_w > f_i)$
            else $f_i' = 0$
        end
4)  Discard all the individuals with fitness value 0
5)  Stop
```

Note:

- Linear scaling (only) may yield some individuals with negative fitness value.
- Hence, Linear scaling is usually adopted after Sigma scaling to avoid the possibility of negative fitness individuals.

Power law scaling

In power law scaling, the scaled fitness value is given by

$$f_i' = f_i^k$$

where k is a problem dependent constant.

Thank You!!

IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

NPTEL

Introduction to Soft Computing

Multi-Objective Optimization

Debasis Samanta

Department of Computer Science and Engineering

IIT KHARAGPUR

Multiobjective optimization problem: MOOP

There are three components in any optimization problem:

F: Objectives

minimize (maximize) $f_i\{x_1, x_2 \dots \dots, x_n\}, i = 1,2, \dots \dots m$

S: Constraints

Subject to $g_j\{x_1, x_2 \dots \dots, x_n\}, ROP_j\ C_j, j = 1,2 \dots \dots l$

V: Design Variables

$X_k ROP_K\ d_k, k = 1,2 \dots \dots n$

Note :

1) For a multi-objective optimization problem (MOOP), m ≥ 2
2) Objective functions can be either minimization, maximization or both.

A formal specification of MOOP

Let us consider, without loss of generality, a multi-objective optimization problem with n decision variables and m objective functions

$$\text{Minimize } y = f(x) = [y_1 \in f_1(x), y_2 \in f_2(x) \dots \dots y_k \in f_k(x)]$$

where

$$x = [x_1, x_2 \dots \dots, x_n] \in X$$

$$y = [y_1, y_2 \dots \dots, y_n] \in Y$$

Here :

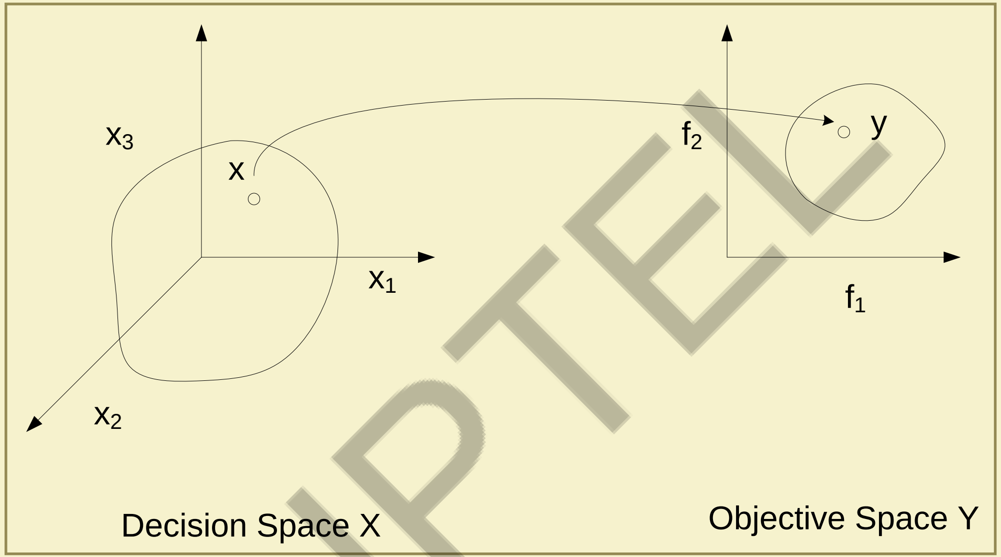
x is called decision vector

y is called an objective vector

X is called a decision space

Y is called an objective space

Illustration: Decision space and objective space



Thus, solving a MOOP implies to search for x in the decision space (X) for an optimum vector (y) in the objective space (Y).

A formal specification of MOOP

In other words,

1) We wish to determine $\bar{X} \in X$ (called feasible region in X) and any point $\bar{x} \in \bar{X}$ (which satisfy all the constraints in MOOP) is called feasible solution.

2) Also, we wish to determine from among the set $\bar{X}$, a particular solution $\bar{x}^*$ that yield the optimum values of the objective functions.

Mathematically,

$$\forall \bar{x} \in \bar{X} \; and \; \exists \, \bar{x}^* \in \bar{X} | f_i(\bar{X}^*) \leq f_i(\bar{X}),$$

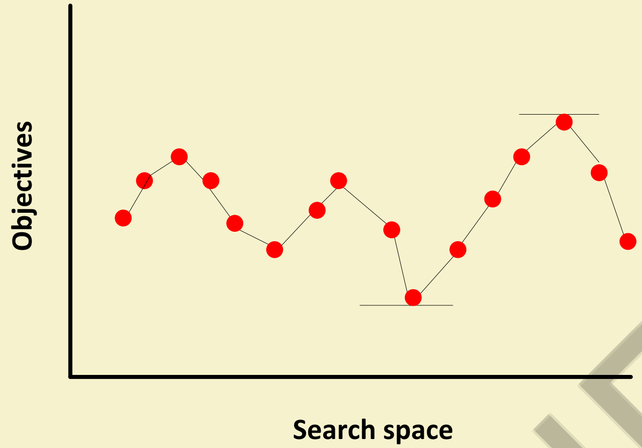
where $\forall \, i \in [1,2, \dots . m]$

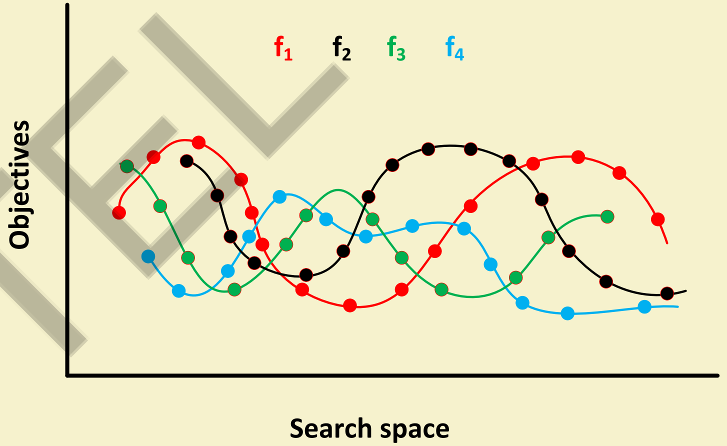
3) If this is the case, then we say that $\bar{x}^*$ is a desirable solution.

Why solving a MOOP is an issue?

- ✓ In a single-objective optimization problem, task is to find typically one solution which optimizes the sole objective function
- ✓ In contrast to single–objective optimization problem, in MOOP:
  1) Cardinality of the optimal set is more than one, that is, there are $m \geq 2$ goals of optimization instead of one
  2) There are $m \geq 2$ different search points (possibly in different decision spaces) corresponding to m objectives
- ✓ Optimizing each objective individually not necessarily gives the optimum solution.
  1) Possible, only if objective functions are independent to their solution spaces.

Illustration: Single vs. multiple objectives

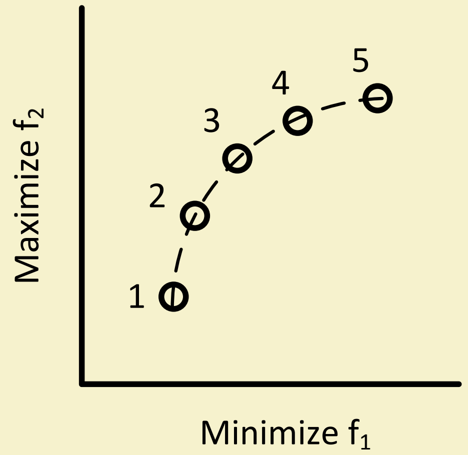




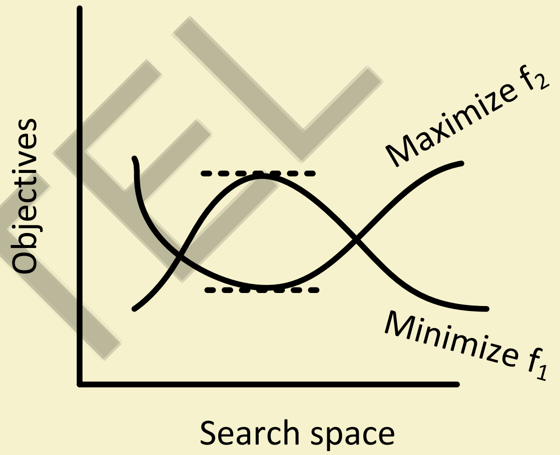
Why solving a MOOP is an issue?

- ✓ In fact, majority of the real-world MOOPs are with a set of trade-off optimal solutions. A set of trade-off optimal solutions is also popularly termed as **Pareto optimal solutions**
  - ➢ In a particular search point, one may be the best whereas other may be the worst
- ✓ Also, sometime MOOPs are with conflicting objectives
  - ➢ Thus, optimizing an objective means compromising other(s) and vice-versa.

MOOP: Trade-off and conflicts in solutions

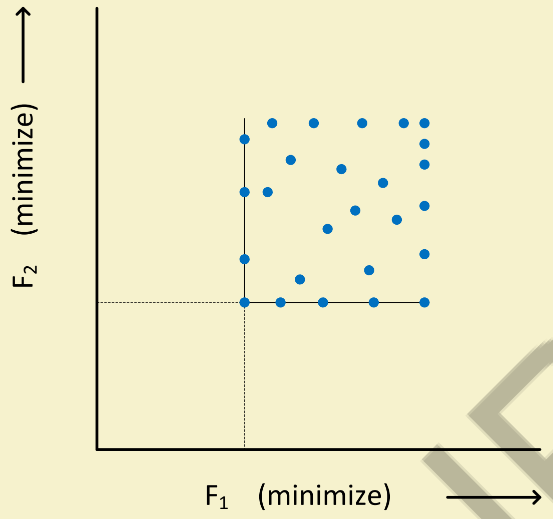


Trade-off optimal solution

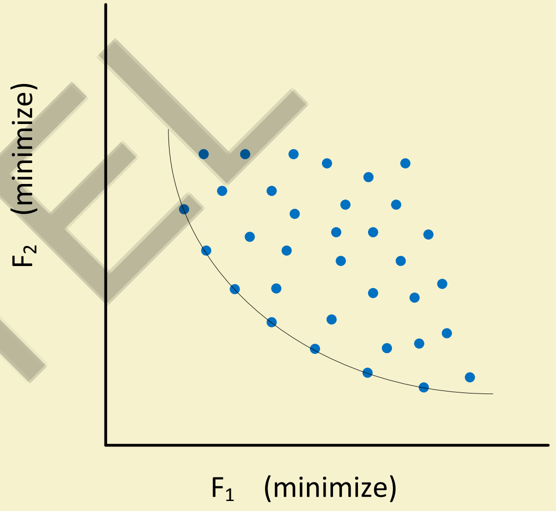


Conflicting objectives

Illustration: ideal solution vs. real solution



Ideal Situation

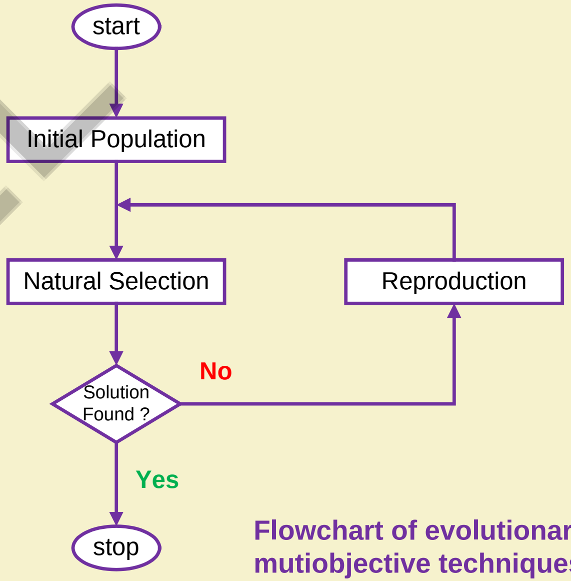


Real Situation

GA-based approach to solve MOOPs

MOEA : Multi Objective Evolutionary Algorithm

- ✓ MOEA follows the same reproduction operation as in GA but follow different selection procedure and fitness assignment strategies.
- ✓ There are also a number of stochastic approaches such as Simulated Annealing (SA), Ant Colony Optimization (ACO), Particle Swam Optimization (PSO), Tabu Search (TS) etc. could be used to solve MOOPs.



Flowchart of evolutionary mutiobjective techniques

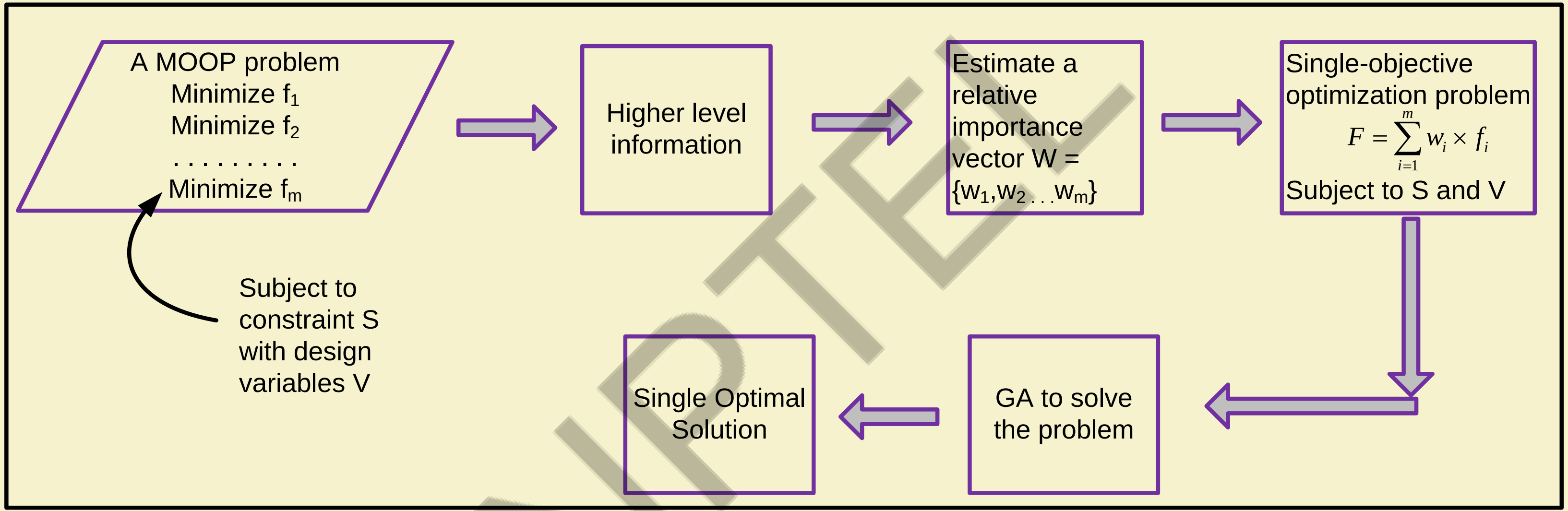
MOEA: GA-based approach to solve MOOP

- ✓ There are two board approaches to solve MOOPs with MOEA
  - ➢ A priori approach (also called preference based approach)
  - ➢ A posteriori approach (does not require any prior knowledge)

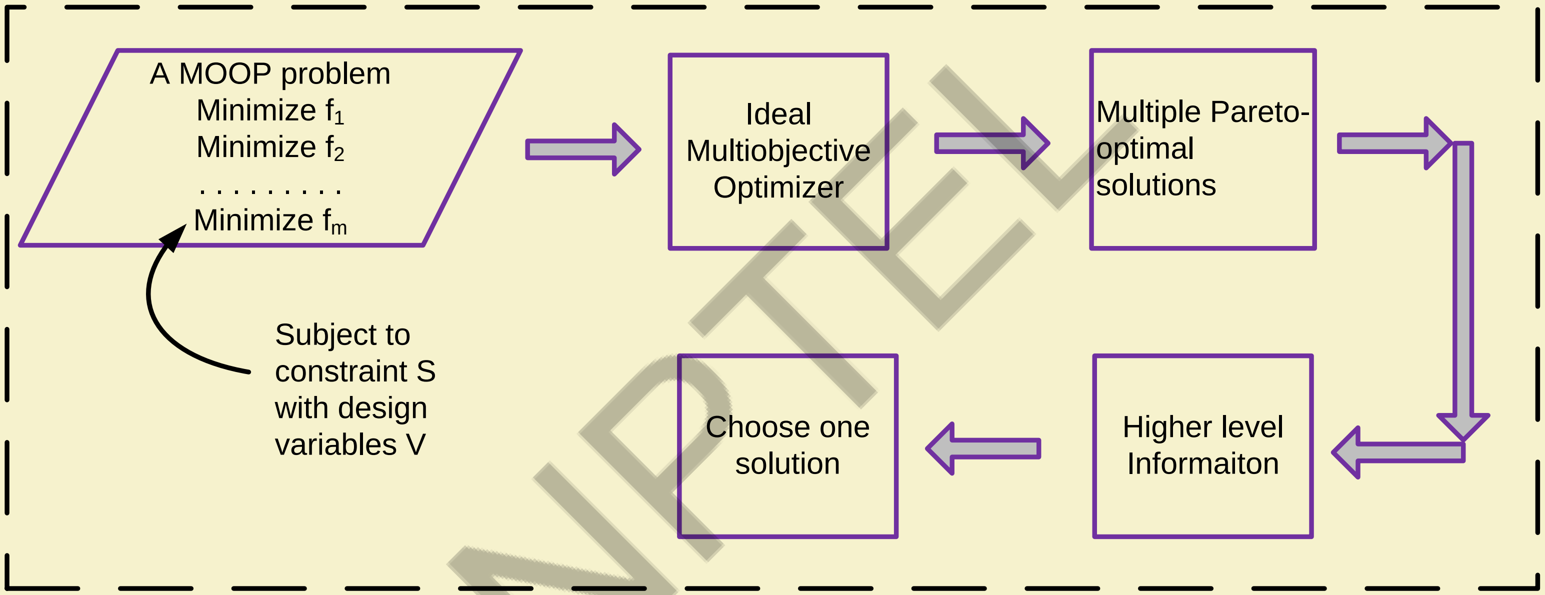
Two major problems must be addressed when a GA is applied to multi-objective optimization problems.

1) How to accomplish fitness assignment and selection in order to guide the search toward the optimal solution set?
2) How to maintain a diverse population in order to prevent premature convergence and achieve a well distributed trade-off front?

Schematic of a priori MOEA approach


A MOOP problem
Minimize f_1
Minimize f_2
. . . .
Minimize f_m
Subject to constraint S with design variables V
Higher level information
Estimate a relative importance vector W = $\{w_1, w_2 \ldots w_m\}$
Single-objective optimization problem
$F = \sum_{i=1}^{m} w_i \times f_i$
Subject to S and V
GA to solve the problem
Single Optimal Solution

Schematic of a posteriori MOEA approach



IDEAL multi-objective optimization

Here, effort have been made in finding the set of trade-off solutions by considering all objective to be important.

Steps

1) Find multiple trade-off optimal solutions with a wide range of values for objectives. (Note: here, we do not use any relative preference vector information). The task here is to find as many different trade-off solutions as possible.

2) Choose one of the obtained solutions using higher level information (i.e. evaluate and compare the obtained trade-off solutions)

Illustration: Higher level information

Consider the decision making involved in buying an automobile car. Consider two objectives.

1) **minimize Cost**
2) **maximize Comfort**

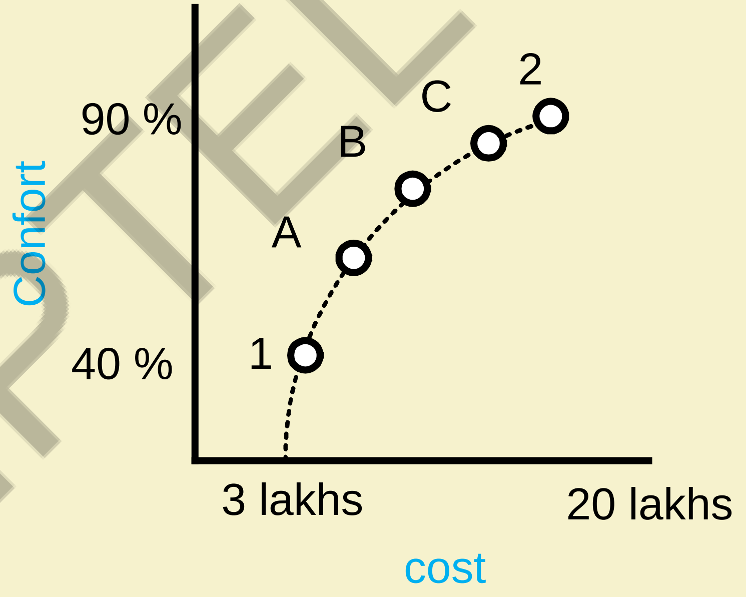


Illustration: Higher level information

- ✓ Here, solution 1 and 2 are two extreme cases.
- ✓ Between these two extreme solutions, there exist many other solutions, where trade-off between cost and comfort exist.
- ✓ In this case, all such trade-off solutions are optimal solutions to a multi-objective optimization problem.
- ✓ Often, such trade-off solution provides a clear front on an objective space plotted with the objective values
- ✓ This front is called Pareto-optimal front and all such trade-off solutions are called Pareto-optimal solutions (after the name of Vilfredo Pareto, 1986)

Choosing a solution with higher level information

- ✓ Knowing the number of solutions that exist in the market with different trade-offs between cost and comfort, which car does one buy?
- ✓ It involves many other considerations
  - ➢ total finance available to buy the car
  - ➢ fuel consumption
  - ➢ depreciation value
  - ➢ road condition
  - ➢ physical health of the passengers
  - ➢ social status
  - ➢ After sales service, vendor’s reputation, manufacturer’s past history etc.

Thank You!!

IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

NPTEL

Introduction to Soft Computing

Multi-Objective Optimization-II

Debasis Samanta

Department of Computer Science and Engineering

IIT KHARAGPUR

Formal specification of MOEA approach

In the next few slides, we shall discuss the idea of solving MOOPs more precisely. Before that, let us familiar to few more basic definitions and terminologies.

1) Solutions with multiple-objectives
2) Concept of domination
3) Properties of dominance relation
4) Pareto-optimization

Solutions with Multiple Objectives

Ideal objective vector

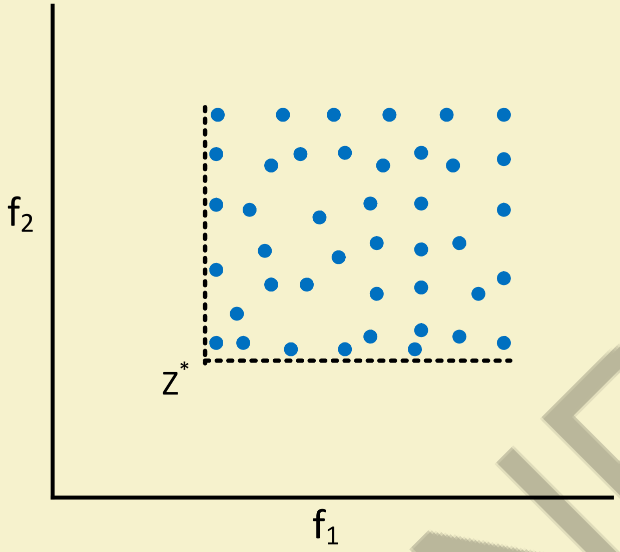
For each of the M^{th} conflicting objectives, there exist one different optimal solution. An objective vector constructed with these individual optimal objective values constitute the ideal objective vector.

Definition 1: Ideal objective vector

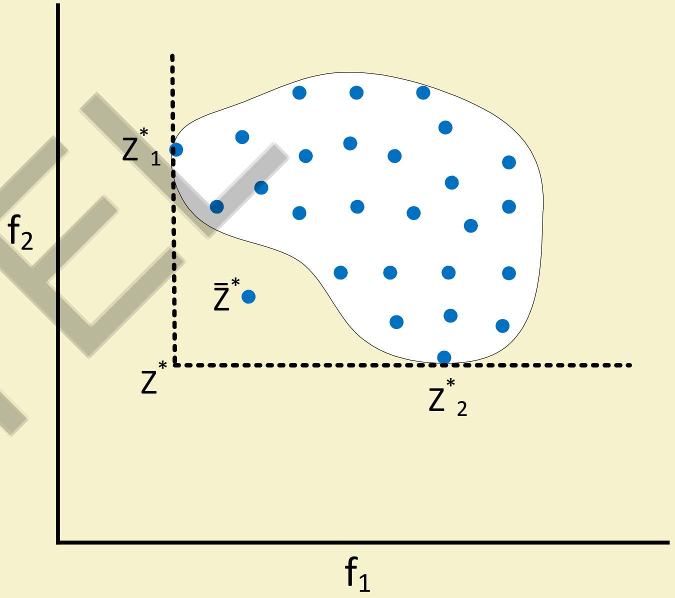
Without any loss of generality, suppose the MOOP is defined as
Minimize $\boldsymbol{f_m}(x), m = 1, 2, \ldots \ldots, M$
Subject to $X \in S$, where S denotes the search space.
and f_m^* denotes the minimum solution for the m^{th} objective functions, then the ideal objective vector can be defined as

$$Z^* = f^* = [f_1^*, f_2^* \ldots \ldots f_M^*]$$

Ideal objective vector : Physical interpretation



(A) Ideal objective vector



(B) A good solution vector should be as close to ideal solution vector

Ideal objective vector : Physical interpretation

Let us consider a MOOP with two objective functions f_1 and f_2 where both are to be minimized.

- ✓ If $z^* = f^* = [f_1^*, f_2^*]$ then both f_1 and f_2 are minimum at $x^* \in S$.
  (That is, there is a feasible solution when the minimum solutions to both the objective functions are identical).
- ✓ In general, the ideal objective vector z^* corresponds to a non-existent solution (this is because the minimum solution for each objective function need not be the same solution).
- ✓ If there exist an ideal objective vector, then the objectives are non-conflicting with each other and the minimum solution to any objective function would be the only optimal solution to the MOOP.

Ideal objective vector : Physical interpretation

Let us consider a MOOP with two objective functions f_1 and f_2 where both are to be minimized.

- ✓ Although, an ideal objective vector is usually non-existing, it is useful in the sense that any solution closer to the ideal objective vector are better.

  (In other words, it provides a knowledge on the lower bound on each objective function to normalize objective values within a common range).

Utopian objective vector

The Utopian objective vector can be formally defined as follows.

Definition 2: Utopian objective vector

A Utopian objective vector Z^{**} has each of its component marginally smaller than that of the ideal objective vector, that is

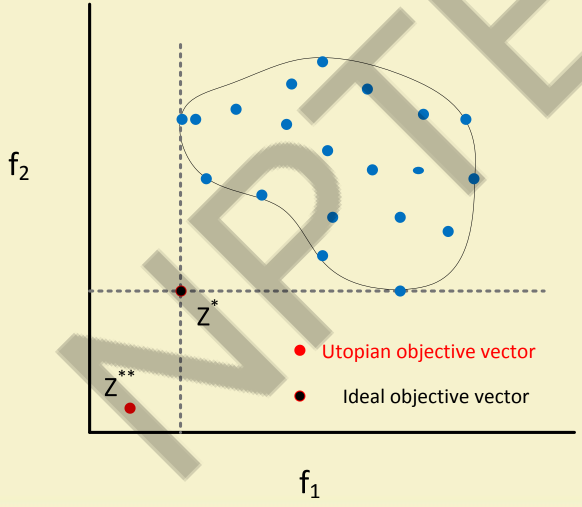
$$Z_i^{**} = Z_i^{*} - \epsilon_i \; with \; \epsilon_i > 0, \forall \, i = 1,2 \ldots \ldots M$$

Note :

Like the ideal objective vector, the Utopian objective vector also represents a non-existent solution.

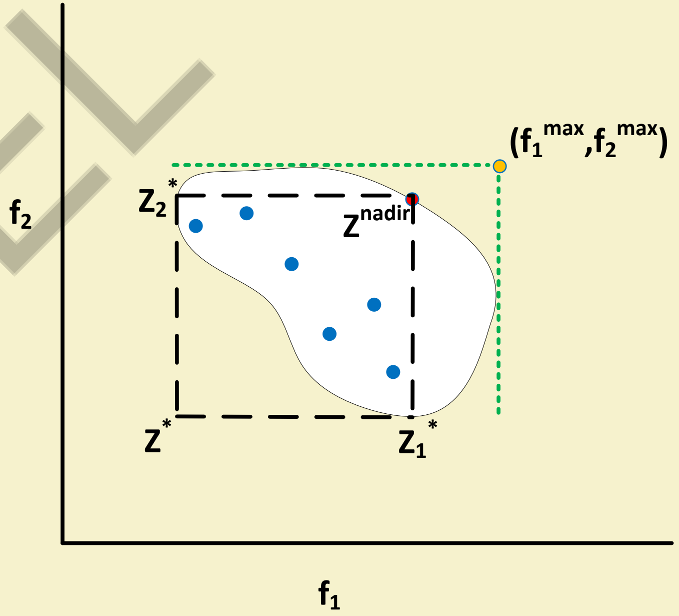
Utopian objective vector

Utopian objective vector corresponding to a solution which has an objective value strictly better than (and not equal to) that of any solution in search space.



Nadir objective vector

The ideal objective vector represents the lower bound of each objective in the entire feasible search space. In contrast to this, the Nadir objective vector, denoted as Z^{nadir} , represents the upper bound of each objective in the entire Pareto-optimal set (note: not in the entire search space).



Nadir objective vector

Note:
Z^{nadir} is the upper bound with respect to Pareto optimal set. Whereas, a vector of objective W found by using the worst feasible function values f_i^{max} in the entire search space.

Usefulness of Nadir objective vector

In order to normalize each objective in the entire range of Pareto-optimal region, the knowledge of Nadir and ideal objective vectors can be used as follows.

$$\overline{f_i} = \frac{f_i - z_i^*}{z_i^{nadir} - z_i^*}$$

Concept of Domination

Concept of domination

Notation

- ✓ Suppose, $f_1, f_2 \dots \dots . . f_M$ are the objective functions
- ✓ x_i and x_j are any two solutions
- ✓ The operator $\lhd$ between two solutions x_i and x_j as $x_i \lhd x_j$ to denote that solution x_i is better than the solution x_j on a particular objective.
- ✓ Alternatively, $x_i \rhd x_j$ for a particular objective implies that solution x_i is worst than the solution x_j on this objective.

Note :

If an objective function is to be minimized, the operator $\lhd$ would mean the "$<$" (less than operator), whereas if the objective function is to be maximized, the operator $\lhd$ would mean the "$>$" (greater than operator).

Concept of domination

Definition 3: Domination

A solution x_i is said to dominate the other solution x_j if both condition I and II are true.

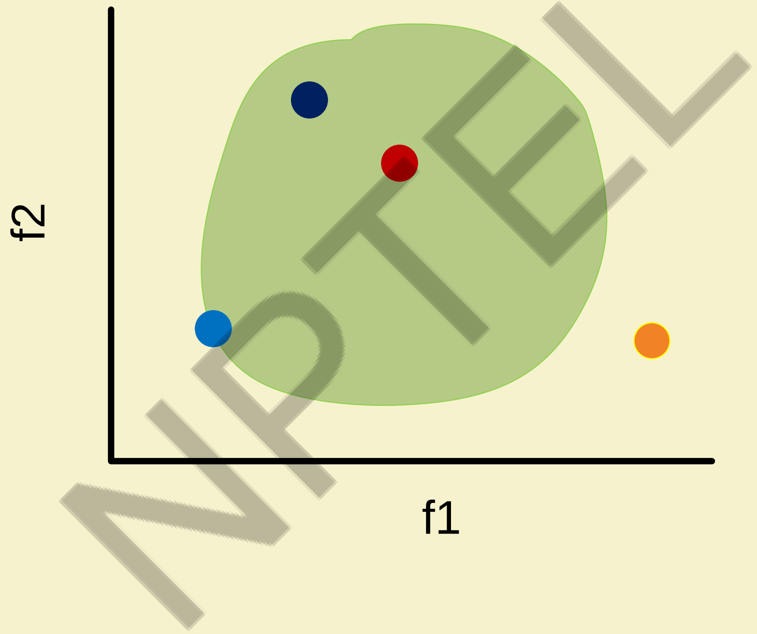
Condition : I

The solution x_i is no worse than x_j in all objectives. That is $f_k(x_i) \ntriangleright f_k(x_j)$ for all $k = 1, 2, \ldots\ldots, M$

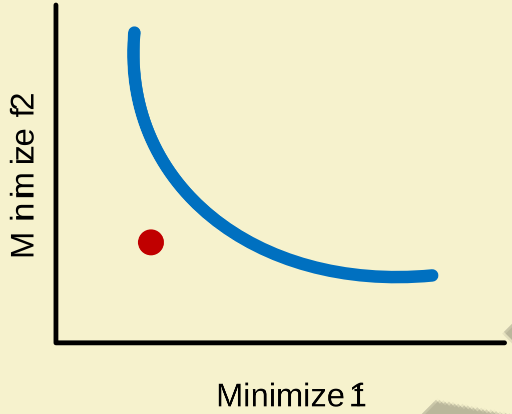
Condition : II

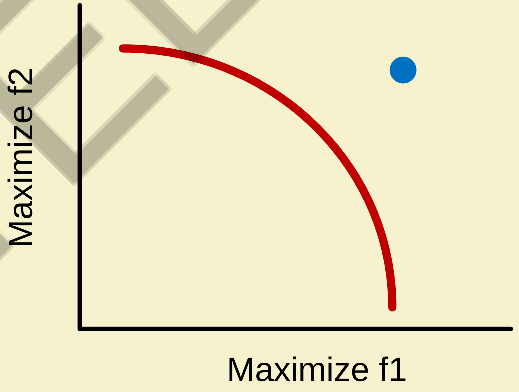
The solution x_i is strictly better than x_j in at least one objective. That is is $f_{\bar{k}}(x_i) \triangleleft f_{\bar{k}}(x_j)$ for at least one $\bar{k} = \{1,2 \ldots\ldots M\}$

Concept of domination

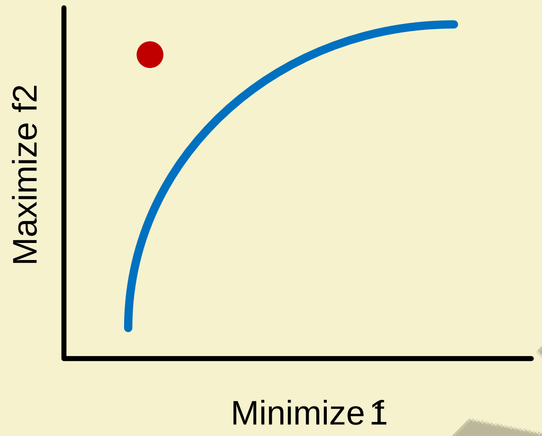


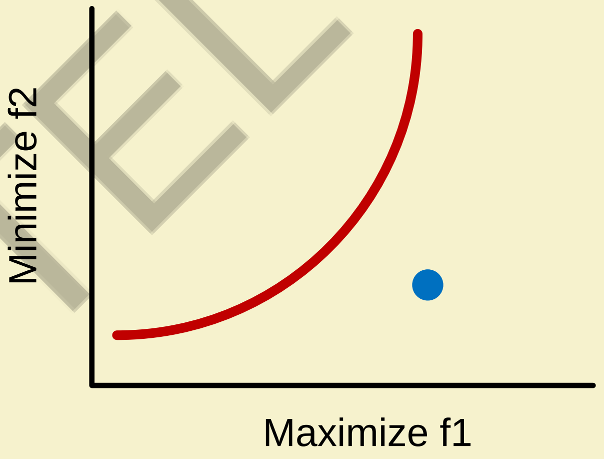
Concept of domination





Concept of domination





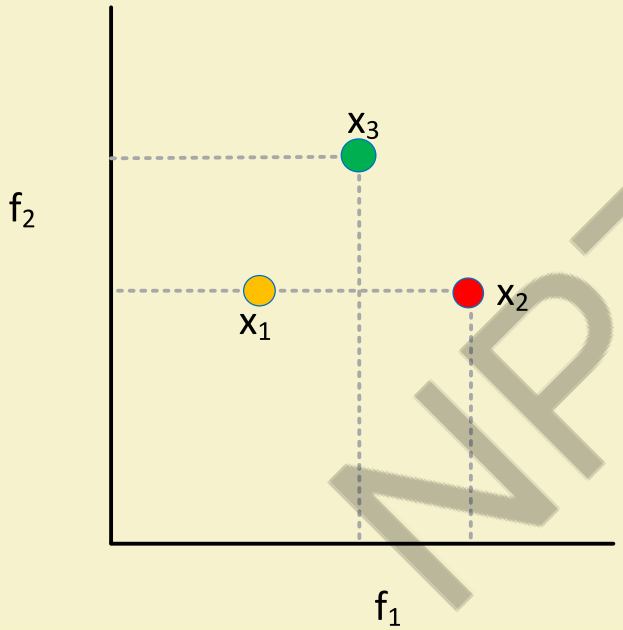
Points to be noted

Note :

- ✓ If either of the condition I and II is violated then the solution x_i does not dominate the solution x_j .
- ✓ If x_i dominates the solution x_j (it is also mathematically denoted as $x_i \leq x_j$.
- ✓ The domination also alternatively can be stated in any of the following ways.
  - ➢ x_j is dominated by x_i
  - ➢ $x_i$is non-dominated by x_j
  - ➢ x_i is non-inferior to x_j

Illustration 1

Consider that $f_1$and f_2 are two objectives to be minimized



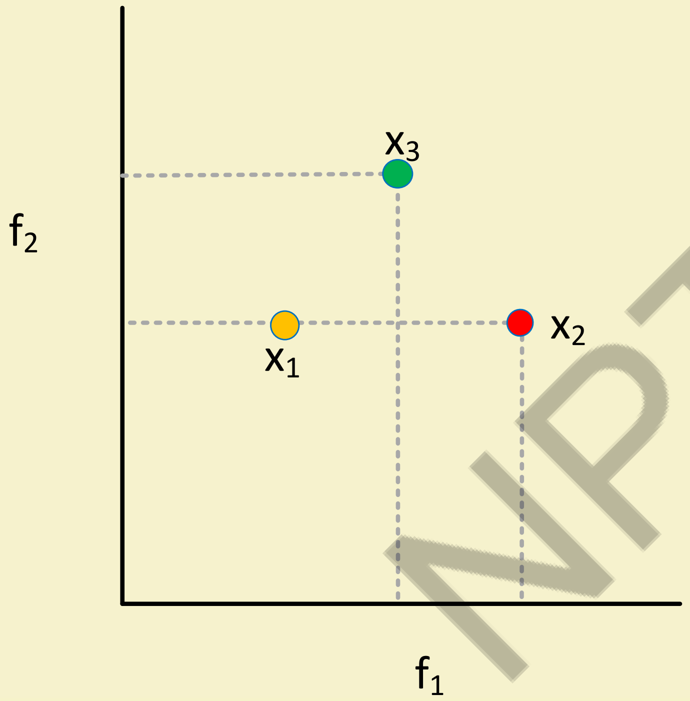
$\boxed{\begin{array}{l}\text{Minimize } f_1 \\ \text{Minimize } f_2\end{array}}$

$x_1 \preceq x_2$

$x_1 \preceq x_3$ but $x_3 \npreceq x_1$

$x_2 \npreceq x_3$ as well as $x_3 \npreceq x_2$

Illustration 2



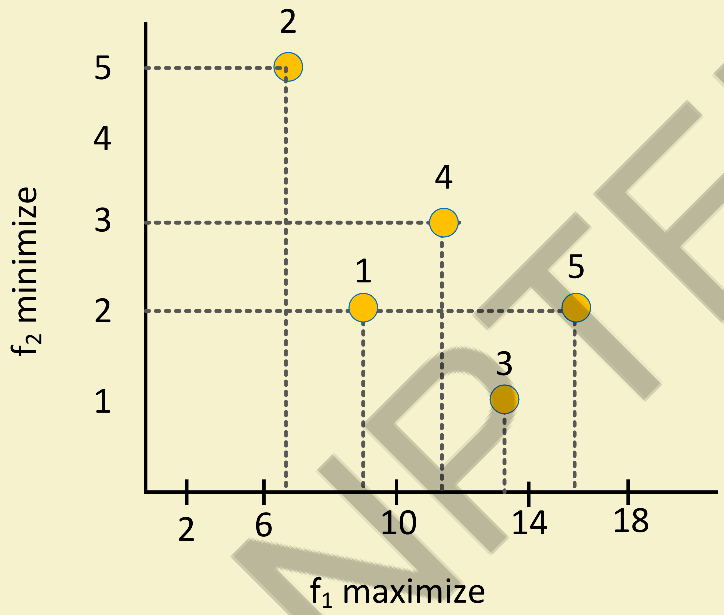
$\boxed{\begin{array}{l}\text{Minimize } f_1\\ \text{Maximize } f_2\end{array}}$

$x_1 \leq x_2$ or $x_2 \leq x_1$?

$x_1 \leq x_3$ or $x_3 \leq x_1$?

$x_2 \leq x_3$ or $x_3 \leq x_2$?

Illustration 3



Here, 1 dominates 2, 5 dominates 1 etc.

Thank You!!