EVOLUTIONARY COMPUTATION (EC)

What is EC?

EC is the subfield of computational intelligence that use ideas and get inspiration from natural evolution.

Examples:-

* Biological evolution of human being
* Foraging and social behavior of swarm to locate food.
* Movement and pattern of ants in search of food

Need:

EC techniques can be used to solve complex optimization problems by generating, evaluating and modifying a population of possible solutions

* EC techniqus for single and multi-objective optimization

Content of the course:

1. Introduction and principles of Evolutionary computation.
2. EC techniques for single objective optimization

* Binary and Real-coded Genetic algorithms
* Differential Evolution

* particle swarm optimization etc

3. Constraint Handling with EC techniques

* KKT conditions
* penalty function Method
* parameter-less Debs Method etc

4. Introduction to multi-objective optimization

* concept of Dominance and pareto-optimality
* Approaches to Multi-objective optimization etc

5 Classical Multi-objective optimization Methods

* Weighted-sum Method
* ϵ- Constraint Method,
* Weighted Metric Methods. etc

6. EC Techniques for multi-objective optimization

* NSGA-II
* SPEA2
* performance Assessment via Indicator

Lec 1: Introduction to optimization

What is search and optimization?

Def: A task of searching for a set of decision variables which would minimize or maximize objective function subjected to satisfying constraints.

Let $f(x,y) = \sin(x)\cos(y)$

* Decision variables = (x, y)
* objective function: $f(x,y)$

Scope of an optimization: Automotive Design

Aim:- Best combination of material with best engineering to provide faster, lighter more fuel efficient and safer vectiles

* For the above aim scientists are used gentic algorithm which gave very good and quicker results. Because it take years to experiment to find the best material.

2. AI Robotics

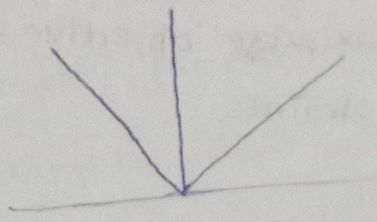
Aim: Intelligence and learning in robotics

Study: Using Genetic Algorithms

Applications:- Robot path planning, Robot vision, Robot Speech, Robot behaviour etc

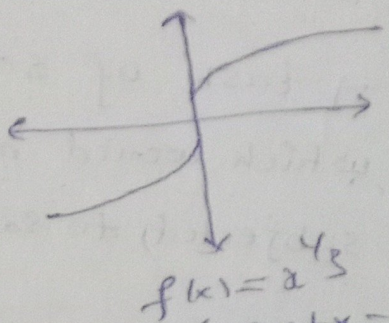
Properties of practical optimization problems

* Non-differentiable functions and constraints



$f(x) = |x|$

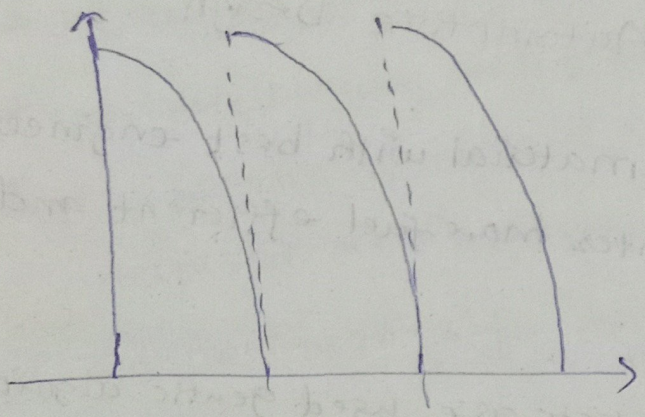
Non-diff at $x = 0$



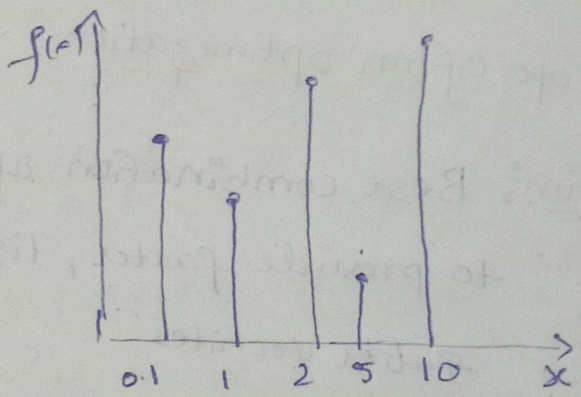
$f(x) = x^{4/5}$

Non-diff at $x = 0$

* Discontineous function



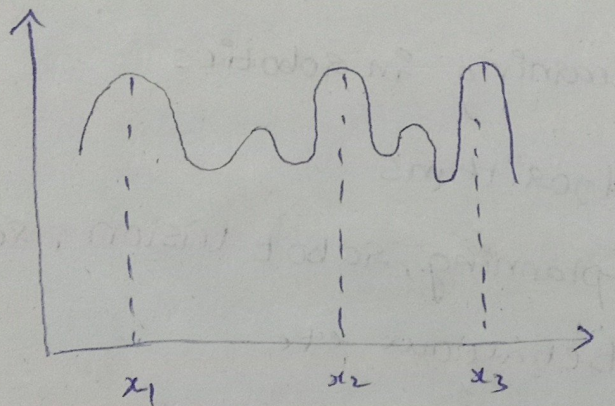
* Discrete/ discontineous search space.



* Mixed variables (discrete, continuous permutation)

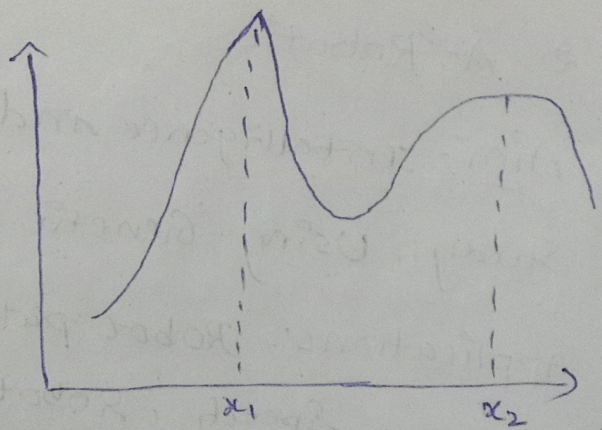
The values of domain are fixed and they are not contineous.

* Multi-modal function



for some problems there will be more than one optimal values but the optimization is not able find all those points.

* Robust solution



x_1 is very sensitive point even if there is small change in it the max value of fn decreases. So we look for robust soln like x_2. But optimization don't know this.

* Non-linear functions (where our optimization fn can stuck in the local minima)
* Computationally expensive prob
  Eg:- topology optimization
* Multi-disciplincay optimization (like we wka take prob of chemical engi, mechanical, Biotech, civil...)

problem formulation :-

A multi-variable single-objective optimizat-on problem is given as

$$\text{Minimize} : f(x)$$
$$\text{subject to } g_j(x) \geq 0 \; ; j = 1, 2, \ldots J$$
$$h_k(x) = 0 \; ; k = 1, 2, \ldots K$$
$$x_i^{(L)} \leq x_i \leq x_i^{(U)} \; ; i = 1, 2, \ldots N$$

* $x = \{x_1, \ldots x_i, \ldots x_N\}^T$ is the decision-variable, vector
* $f(x)$ is the objective function (which we have to minimize)
* $g_j(x)$ is the j-th inequality constraint
* $h_k(x)$ is the k-th equality constraint
* $x_i^{(L)}$ and $x_i^{(U)}$ are the lower bound and upper bound on the i-th decision variable

* Need for optimization
* problem formulation or modeling
  * Identify problem parameters
  * Choose design variables (parameters which change on time) from parameters
  * Formulate constraints (conditions)
  * Formulate objective function
  * set up variable bounds
  * Requires 50-60% of the effort
* Choose an optimization algorithm
* obtain solution
* Reformulation and rerun if desired

Design Variables:-

* A design problem usually involves many design parameters
  * List any and every parameter related to the problem
* parameters sensitive to the given algorithm design or problem can be considered as design variables in the parlance of optimization procedure
  * Sensitivity analysis, etc
  * Expaience of the users can be used
* Specify the type of each parameter (binary, discrete, real)

* First thumb rule of an optimization problem:
choose a few variables as possible
* Efficiency and speed of optimization algorithm depend, to a large extent on the number of chosen design variables

Constraints:- It represent limit on certain resource or on certain physical phenomenon, for example, satisfy stress limitation, current or voltage restriction .etc

* Identify the constraints associated with the optimization problem.

Types of constraints :-

* Inequality constraint: $g(x) \geq 0$ or $g(x) \leq 0$
  * Mostly encounter in engineering design problems.
* Equality constraint:- $h(x) = 0$, difficult to handle
* Handling of equality constraint, for example, deflection of beam, say $\delta(x) = 0.35$ mm, can be converted it into inequality, say

$\delta(x) \geq 0.25$ and $\delta(x) \leq 0.45$ (because the range of values is easy to get optimal value)

* Second thumb rule in the formulation of optimization problem : The number of complex equality constraints should be kept as low as possible.

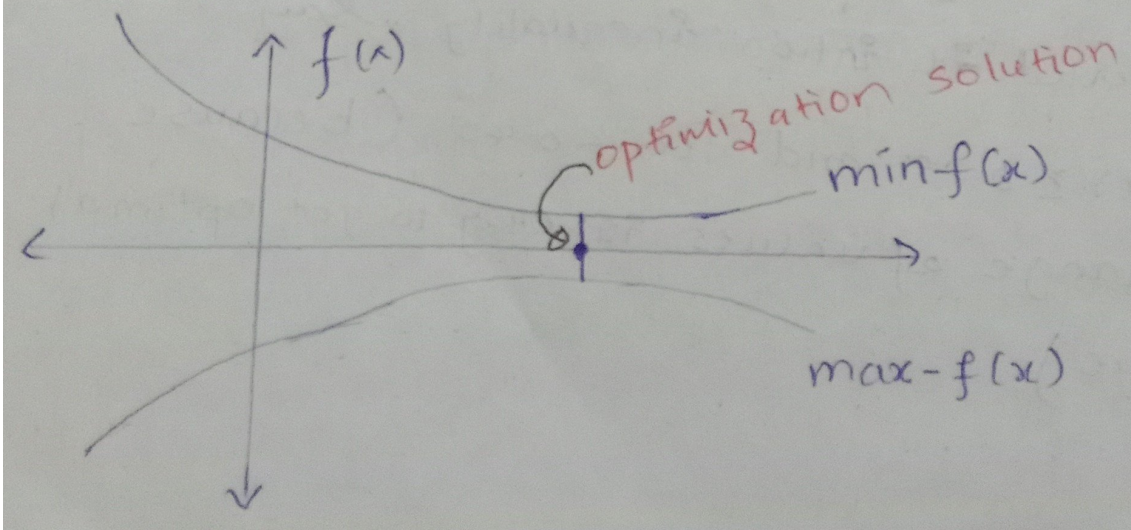
Objective Function :-

* Minimize or Maximize $f(x)$, which is written in terms of design variables and other parameters
* optimization algorithms usually written either for minization or maximization
* Let's suppose in your organization you have only access to minimization algorithm but you want to maximize the function "$f(x)$". But yo can you maximize $f(x)$ using it ?

Ans :- yes, How ?

* Duality principle helps unification.

$$\min f(x) = \max F(x) \text{ where } F(x) = -f(x)$$



Optimization Algorithm:

It provides systematic and efficient ways of creating and comparing new design solutions in order to achieve an optimized design solution.

* optimization algorithm works on a mathema -tical model of the optimal design problem.
* Time consuming and computationally expen -sive prodecure because it requires compar -ison of a number of design variables.

Critical Remarks on Numerical optimization Techniques:

* One method is not applicable in many optimization problems.
* Constrined handling is sensitive to penalty paramet
* Not efficient in handling discrete variables
* Local perspective of searching
* uncertainties in decision and state variables
* Noisy / dynamic optimization problems
* Multiple objectives optimization problems

Need for an innovative and flexible optimization algorithm.

* That need is fullfilled using EC techniques

Evolutionary Computation:-

Introduction:

* Nature inspired algorithm (but not copied)
    * Mimic natural or biological phenomena or process.
* Based on natural evolution + genetics

  Natural evolution: offspring (new solutions) are created by variation operators, such as crossover, mutation etc

  Survival of the fittest: Good solutions are retained and bad are deleted.

  Genetics: Information is coded.
* Evolutionary computation (EC) techniques can be used in optimization, learning and design
* EC techniques are population-based algorithms.
    * Many solutions or points are considered in the process of optimization then finally converge to a optimal point. So the multiple solutions are called as population.

* Nature as structural

Nature as an optimizer :-

* Nature as structural engineer
  * Stem, Bamboo, insect trachea, bee-hive
* Nature as a computational fluid dynamics (CFD) solver:
  * Birds, fishes
* Nature as a drag reducer
  * penguin body

Genetics

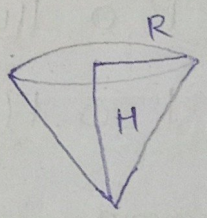
Genotype → internally stored [coded] in genes

Phenotype → physical appearance (structure)

cone-1 (binary coded)

Chromosome = {01001 01010}

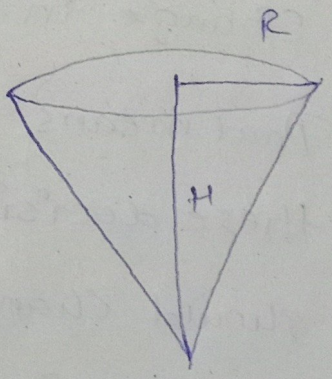
(R, H) = (9, 10) cm



cone-2

Chromosome = {01100 01111}

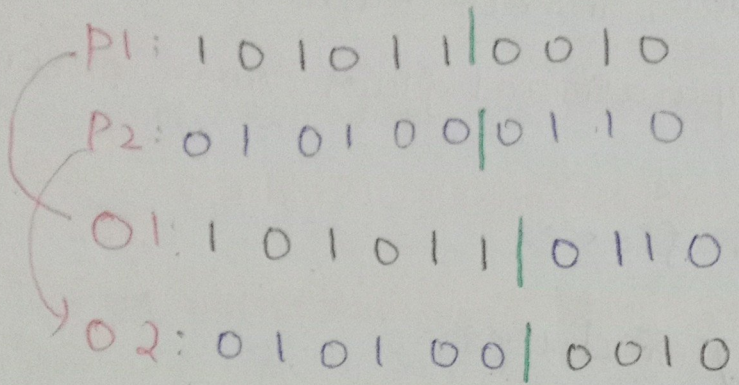
(R, H) = (12, 15) cm



* That's why genetics stores information internally and phenotype tells us how it look like.

Natural Evolution :-

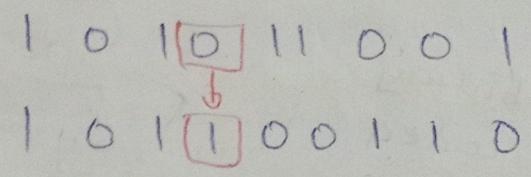
* Crossover between two parents at the random site (say 6th site)



} Means when we changed the tail

which implies that the radius or height of the cone will change in phenotype

* Mutation at the random bit position (say 4th)



Means if we change one binary code, R or H will change in cone.

* That means when we perform variation operator these decision variables or design variables should change when we perform either cross-over or mutation

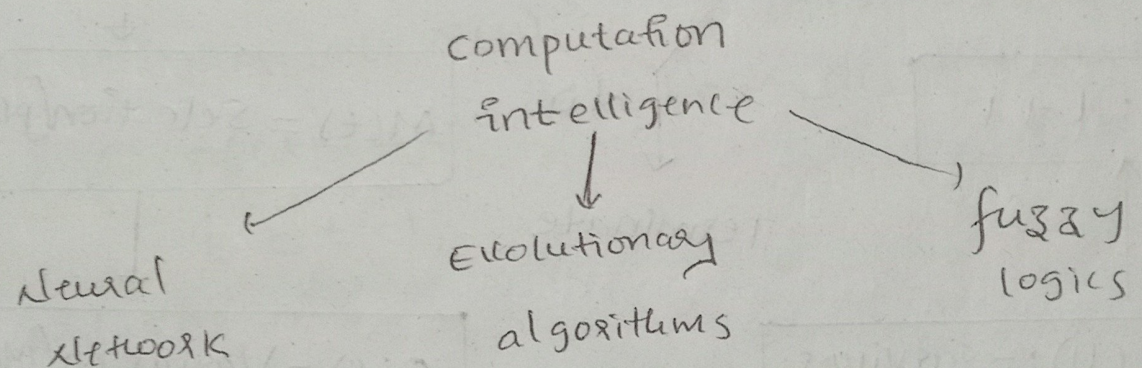
* Darwin's principle of evolution:

* Survival of the fittest:- stronger candidate

has more chance of survival than weaker candidates in an environment of limited sources like food, water etc.

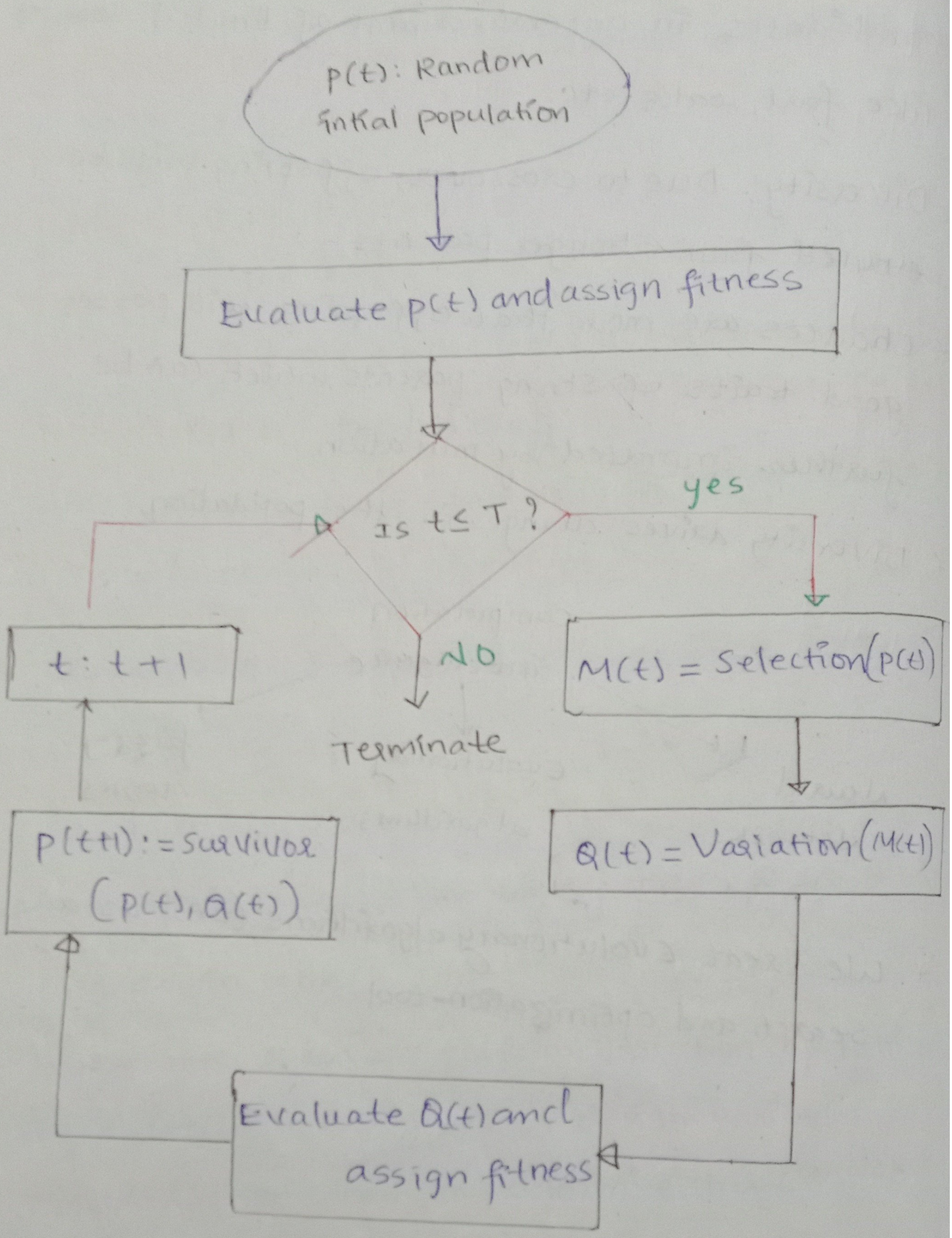
Diversity:- Due to crossover, offspring will be evolved from stronger parents

* chances are more that offspring will posses good traits of strong parents which can be further improved by mutation.
* Diversity drives changes in the population.



* We treat Evolutionary algorithms (EA) as a search and optimization tool.

Flow-chart of EC Algorithms



Generalized framework of EC Algorithms

Algorithm - 1

1. Solution representation
2. Input: t=1 (Generation counter),
   Maximum allowed generation = T
3. Intilize random population (p(t));
4. Evaluate (p(t)); %. Evaluate objective, constrain
   -ts and assign fitness.
5. while t ≤ T do
   M(t) := Selection(p(t)); %. survival of the fittes
   Q(t) := Variation (M(t)); %. Crossover & Mutation
   Evaluate Q(t)); %. Offspring population
   p(t+1) := Survivor (p(t), Q(t)); %. survival of fittest
   t := t+1;
   end while

Advantages of EC Techniques:

* Applicable in problems where no (good) method is available.
  * Multi-modalities, disconti, non-linear problems
  * Noisy problems
  * Implicitly defined problems (simulation models)
  * Discrete variable space.
* Most suitable in problems where multiple -Solu -tions are sought.

* Multi-modal optimization problems
* Multi-objective optimization problems

* No preassumptions with respect to problem space
* Low development costs, i.e costs to adapt to new problem spaces
* parallel implentation is easier for computationally expensive problems

Dis-advantages of EC

* No ~~quantr~~ guarantee for finding optimal solutions in a finite amount of time
  * however asymptotic convergence proofs are avaliable.
* parameter tuning mostly by trial-and-error
  * self-adaption is a remedy
* population approach may be expensive
  * parallel implementation is a remedy

Differences with Numerical optimization Tech

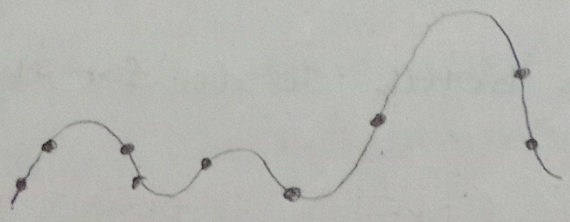
* Different types of variables can be used with Ec techniques
  * A problem requires mixed-variable can be handled.
  * variables such as pamutation, contineous, discrete can be used together

* EC-techniques are population-based metaheuristics
  * The number of solutions can provide safety against premature convergence that can lead to a global perspective to EC-techniques.
  * Implicit parallism
* operators of EC-tech are probabilistic in nature
  * It reduces the chance of getting stuck at the local optima.
* EC-techniques do not require any gradient information
* EC-tech are ideal for parallel computation.
  * Evaluations can be distributed the computing processors and threads

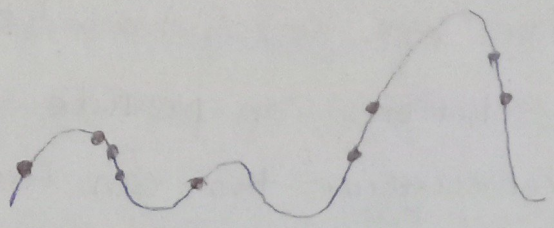
Typical Behaviour of EC-Tech or EA

phases in optimization on a 1-D fitness landscape.

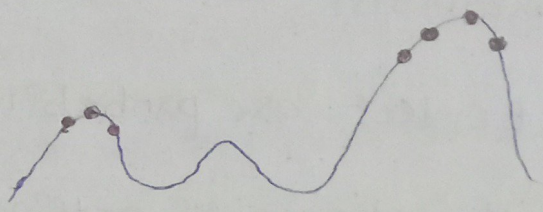
* Early phase: quasi random population distribution



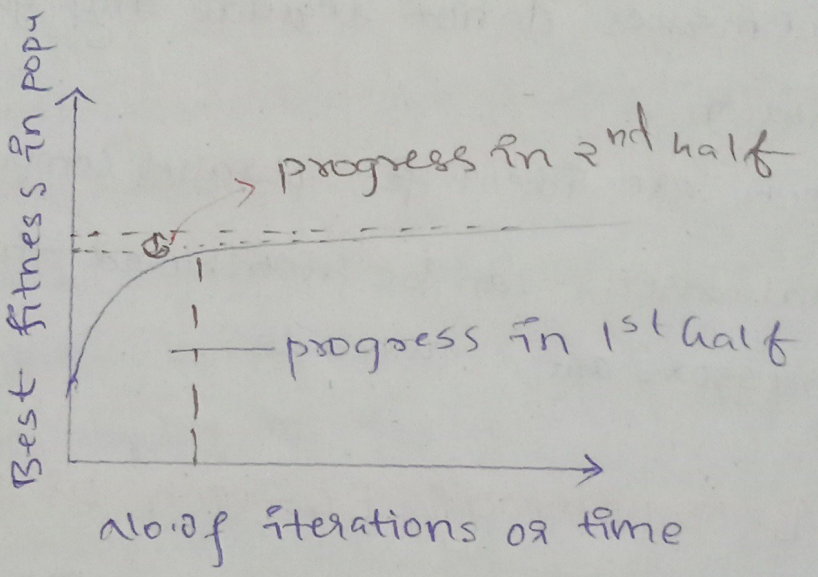
* Mid phase: population arranged around / on hills
  on applying variations



* Late phase: population concentrated on hills.



Typical Run: Convergence plot



Are long runs beneficial?

Ans! * It depends on how much precision/ time you want to the last bit of progress

* It may be better to do for short time or run.

No free Lunch (NFL) Theorm for optimization

* In the context of optimization.

* A framework is developed to explore the connection between effective optimization algorithms and the problems they are solving.
* For any algorithm, any elevated performance over one class of problems is offset by performance over another class.
* Algorithms A1 and A2
* All possible problems F in one class
* performances P_1 and P_2 using A1 and A2 for a fixed number of evaluations.
* $P_1 = P_2$
* NFL breaks down for a narrow class of problems or algorithms.

Lec-3

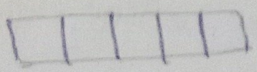
Binary-coded Genetic algorithm (BGA)

Solution Representation:-

Decision variables for an optimization problem are represented using Boolean variables in BGA.

* Binary variable: {0, 1}
* Real variable
* Discrete and integer variable

Real Variable (x_i)



* suppose string length (l) = 5 is chosen for x_i
* maximum value of binary string of (l) = 5, that is, DV(s) of {11111}$_b$ = b = 31 and minimum value is a = 0
* suppose lower bound is $x_i^{(L)} = 3$ and upper bound $x_i^{(U)} = 10$
* value of decision or design variable,

$$x_i = x_i^{(L)} + \frac{x_i^{(U)} - x_i^{(L)}}{2^l - 1} DV(s)$$

DV(s) is a decoded value of binary string

$$\Rightarrow x_i = 3 + \frac{7}{32-1} DV(s) = 3 + \frac{7}{31} DV(s)$$

* DV(s) = (string is = {11011}
* Decoded value (s) DV(s) =

$$(1\times2^0) + (1\times2^1) + (0\times2^2) + (1\times2^3) + (1\times2^4)$$

$$= 1 + 2 + 8 + 16 = 27_{11}$$

* $x_i = 3 + \frac{7}{31} \times 27 = 9.096$
* precision = $\frac{7}{31} = 0.226$
* The neighbour of 9.096 is 9.096 + 0.226 = 9.322
* We cannot get any value of x_i b/w 9.096 & 9.322

⇒ The Real variable x_i is discrete (when we are converting binary string to real variable the default decision variables are discrete).

* If we want a precision of 0.01, then $\frac{7}{2^l - 1} = 0.01$

$\Rightarrow l = \log_2(1 + 7/0.01) = 9.453$ or 10

Working principles through an Example:-

Rosenbrock function:

Minimize $f(x_1, x_2) = 100(x_2 - x_1^2) + (1 - x_1)^2$, bounds

$-5 \leq x_1 \leq 5$ and $-4 \leq x_2 \leq 4$

* optimum soln is $x^* = (1,1)^T$ and $f(x^*) = 0$

Solution Representation:

* Let the chromosome string length is $l = 10$
  * First five bits are used to represent x_1 and rest of them for x_2

Generate intial random population:

* Let the population size is $N = 8$

Index	chromosomes	DV(x_1)	DV(x_2)
1	01100 11010	12	26
2	11000 01011	24	11
8	11100 11000	28	24

* The scaling formula is

$$x_1 = x_1^{(L)} + \frac{x_1^{(U)} - x_1^{(L)}}{2^4 - 1} DV(s) \qquad x_2 = " = 4 + \frac{8}{2^5 - 1} (DV(s))$$

⇒ The decoded value of 01100 is

$$DV(x_1^{(1)}) = 12 \qquad DV(x_2^{(1)}) = 26$$

$$\Rightarrow x_1^{(1)} = -5 + \frac{10}{2^5 - 1} 12 = -1.129 \qquad \Rightarrow x_2^{(1)} = -4 + \frac{8}{31} 26 = 2.7$$

Solution 1

Let soln 1 is represented as $x^{(1)} = (-1.129, 2.710)^T$

⇒ The objective fn value:

$$f(-1.129, 2.710) = 100(2.710 - (-1.129)^2)^2 + (1 - (-1.129))^2$$

$$= 210.455$$

* Let us choose the fitness value same as the function value

Index	chromosomes	$DV(x_1)$	$DV(x_2)$	x_1	x_2	$f(x_1, x_2)$
1	01100	11010	12	26	-1.129	2.710	210.445
2.	11000	01011	24	11	2.742	-1.61	7536.407
⋮						
8	11100	11000	28	24	4.032	2.194	19793.185

Termination condition

* BGA gets terminated when generation counter is more than the allowed maximum generations, that is, $(t > T)$
* Some other criterion can also be considered for terminating the algorithm.
* Since it is the 1st generation we continue and move to selection operator.

Selection

* The purpose is to identify good (usually above avg) solutions in the population.
* In this process we eliminate bad solutions in the population
* we make multiple copies of good solutions
* Selection can be used either before or after search/variation operators.
* when selection is used before search/variation operators, it is called as reproduction or selection
* After search/variation operators: The process of choosing the next generation population from parent and offspring populations is referred to as survivor or elimination. It is also bring referred to as environmental selection.

* Common selection operators are;
  * Fitness proportinate selection
  * Tournament selection
  * Ranking selection, etc.

Binary Tournament selection operator;

* It is similar to playing a tournament among the teams, Here, binary stands for tournament between two teams
* Outcome of binary tournament selection operator is: 'win', 'loss', or 'tie'.
* It is performed by picking two solutions randomly and choose the one that has better fitness value

Index	1	2	3	4	5	6	7	8
fitness	210 .445	7736 .407	14041 .832	1546 403	189. 732	671. 924	7552 -209	17793. 183

** performing without replacement

Index	$f(x_1, x_2)$	winner
2	7536.407	index 4
4	1546.603	

Index	$f(x_1, x_2)$	winner
5	189.732	Index 5
7	7552.209	

7 - Index 7
3 -

9 - Index 4
2 -

8 - Index 6
6 -

3 - Index 6
6 -

1 - Index 5
5 -

8 - Index 1
1 -

observations: winner is selected based on minimum fitness value.

* we can see index 5 have minimum value among all, so it has two copies in the table.
* we considered only two tournaments because population size N = 8;
* We say stochastic in nature when we pick the solutions randomly

Crossover operator:-

* crossover operator is responsible for creating new solutions. These new solutions explore the search space.
* Crossover is performed with probability (P_c). Generally, the P_c is kept high that supports exploration of search space.
* Types of cross-over operators:
  * single-point operator
  * n-point cross-over operator
  * Uniform crossover operator
* Let's choose single-point operator for this example.

Single-point Crossover operator:- For performing single point crossover, two solutions are picked randomly (without replacement) from the pool at a time.

Mating-pool :- is created after performing selection operator.

	old index	new index
Tournament-1	4	1
	7	2
	6	3
	5	4
T-2	5	5
	4	6
	6	7
	1	8

* Let solutions with the following new indexes are picked for performing cross-over.

* solutions {4,7}, {8,2}, {5,1}, and {6,3}

* The random numbers are generated for each pair of solutions. These random numbers are compared with P_c for performing cross over.

* Let $P_c = 0.9$

pair	Random number (r)
{4,7}	0.75
{5,1}	0.93
{8,2}	0.23
{6,3}	0.68

* For the first pair, since $r = 0.75 < P_c = 0.93$, we perform crossover operator.
* Randomly, we choose 8th site to perform cross-over.

Index	Chromosome	DV(s)	$f(x_1, x_2)$
P4:	10100111|01	(20,29)	189.732
P7:	01101010|00	(13,8)	67.124
O4:	10100111|00	(20,28)	125.336
O7:	01101010|01	(13,9)	545.121

11ly we crossover other index pairs by choosing random site

* We observe that cross-over can create good or bad solutions with respect to their parent solutions.
* If a bad solution is created, it will be eliminated during selection in further generations.
* If a good solution is created, it will have multiple copies in further generations.
* As cross-over is performed on two parent solutions which survived the tournament selection
* New solutions / offsprings will more likely to preserve good traits of parents and will evolve as better solutions than their parents

Offspring population after Crossover

New Indexes	$f(x_1, x_2)$	Chromosomes
4	125.336 (04)	1010011100
7	545.121 (07)	0110101001
8	540.287	0110111011
2	12236.916	0010011010
5	189732	1010011101
1	15.46.0	0100610111
6	1351.054	0100011000
3	812.047	0110100111

** when we are using BGA we actually don't need $f(x_1, x_2)$, D(V(s), x_1 x_2 colums only we need 'index' and 'chromosomes' columns

Mutation :-

* Mutation operator is also responsible for search aspect of GA
* The purpose of mutation is to keep diversity in the population.
* Mutation is generally performed with a small probability (P_m)

Bit-wise mutation operator:-

* A solution is chosen with the probability (P_m = 0.10) and a bit is chosen for mutation.
* following are the random numbers for performing mutation.

Index	Random number (r)
1	0.05
2	0.32
3	0.15
4	0.01
5	0.06
6	0.24
7	0.5
8	0.54

* For soln 1, since $r \leq 0.05 < P_m = 0.1$ we perform mutation
  ⇒ Randomly, we pick 4th bit to mutate 0 to 1 or 1 to 0

Index	chromosomes	$f(x_1, x_2)$
1	0 1 0 0 0 1 0 1 1 1	153 46.603
1	0 1 0 1 0 1 0 1 1 1	154.658 good

* For soln-2: $r = 0.32 > P_m = 0.1$, we donot perform mutation

Index	chromosomes	$f(x_1, x_2)$
2	0 0 1 0 0 1 1 0 1 0	12236.916

11th soln 5 $r = 0.15 > P_m = 0.1$, we don't perform

For soln 4 $r < P_m$, so we perform mutation

Index	chromosomes	$f(x_1, x_2)$
4	10101[0]11100	125.336
4	1010[1]11100	1208 very good

* For soln 5, since $r = 0.06 < P_m = 0.1$

Randomly pick 1st bit for mutation

Index	chromosomes	$f(x_1, x_2)$
5	[1]0100111101	189.732
5	[ ]0100111101	10585.571

very bad.

* For other soln, since $r > P_m$ we don't perform

offspring population after mutation

Index	chromosomes	$f(x_1, x_2)$
1	0101010111	154.658
2	0010011010	-12236.916
3	0110100111	812.047
4	1010111100	1.208
5	0010011101	10585.571
6	0100011000	1351.054
7	0110101001	545
8	0110111011	540.287

* similar to cross-over operator, mutation operator can create better or worse solutions than parent solution.
* However, good solutions will be emphasized and bad solution may get deleted by selection operation in further generations.

Last stage: **Survivor**

* It is used to preserve good solutions for the next generation
* Survivor or elimination stage is also referred to as environmental selection.

$(\mu + \lambda)$ – strategy

* μ stands for parent population and λ stands for offspring population.
* We combine both the population and choose the best solutions for the next generation population.

parent population		offspring population	
Index	$f(x_1, x_2)$	Index	$f(x_1, x_2)$
P1	210445	01	154
P2	7536.409	02	12236.916
P3	14041.882	03	812
P4	15346.603	04	1.208
P5	189.732	05	10555.054
P6	671.929	06	1851
P7	9252.201	07	545
P8	19793.183	08	540.287

* Since it is the minimization problem, we select solutions in an ascending order of their fitness value

Remember; in this example we took function output value as fitness value.

* Select 04 01 P5 P1 08 07 P6 and 03 as we have population size N=8,

Next generation population

Index	Chromosomes	$f(x_1, x_2)$
1	1010111010	1.208
2	0101010111	154.653
3	1010011101	189.732
4	0110011010	210.445
5	0110111011	540.237
6	0110101001	545.121
7	0110101000	671.925
8	0101001111	812.043

all are in ascending order

LEC-4: operators and simulations of Binary-coded Genetic Algorithm.

Selection operators: 2. Fitness proportionate Selection Operator. (FPS)

* probability of selecting individual i from population P of size N is: $P_i = \frac{f_i}{\sum_{j=1}^{N} f_j}$ where,

f_i is the fitness value

* It is suitable for maximization problem.
* It is known as roulette wheel selection method.

Index	$f(x_1, x_2)$	P_i	P_i (cumulative)
1	210.445	0.0041	0.0041 ← 0.07128
2	75364	0.147	0.1512 (0.0041 + 0.147)
3	14041.8	0.274	0.4252
4	1546.6	0.030	0.4552
5	189.7	0.0037	0.4591
6	671.9	0.0131	:
7	7552.2	0.1415	:
8	19793.1	0.3863	:
Sum	51242.386		

Calculation

$P_i = \frac{0.04 + 210.445}{51242.386}$

$= 0.0041$

All others are calculated

P_i → Cumulative probability.

Table-(1)

* r_p is the random number b/w 0 to 1

Let $r_1 = 0.07218$. As per cumulative probability Soln 2 is selected. Bcz 0.07218 present b/w 0.0041 & 0.1512)

* Let $r_2 = 0.65799$, Soln 8 is selected.

* The rest of the random numbers are

$r_3 = 0.449176$ $r_5 =$ — $r_8 =$ —

$r_4 = 0.311472$ $r_6 =$ —

$r_7 =$ —

Selected solutions: 7, 5, 7, 7, 8, 8

∴

Index	$f(x_1, x_2)$
2	7536.4
3	14041.8
7	7252.2
7	7252.2
7	7252.2
5	19793.1
5	19793.1
5	19793.1

* Two solutions become "super solution" in population

2.2

8. Stochastic Remainder Roulette-Wheel selection operator

* we calculate probability of each solution using the formula used with fitness proportionate selection operator
* we then multiply the population size (N) with the probability of each solution.

Index	$f(x_1, x_2)$	P_i	NP_i
1	210.445	0.0041	$8 \times 0.0041 = 0.0329$
2		0.1471	1.1966
3		0.274	2.1922
4			
5			0.2015
6			0.0296
7			0.1049
8			1.1322
			3.0901

* we assign a number of copies to the solution based on the integer value of product NP_i (Table 2)

* For ex. $NP_1 = 0.0329$ for soln 1, so we do not assign any copy to it

* for soln 2, $NP_2 = 0.17$, ∴ we assign 1 copy of it

* ∴ Total number of solutions selected is 7 (6+1+stotale (+3)

but we have to choose 1 more soln to keep the population size fixed, that is $N = 8$

* We then use the fitness proportinate selection operator using only the decimal values

Index	NPi	Decimals	Pi (cummulative)
1	0.0329	0.0329	0.0329
2	1.1766	0.1766	0.2094
3	2.1922	0.1922	0.4017
4	0.2415	0.2415	0.6431
5	0.0296	0.0296	0.6728 ← 0.763355
6	0.1049	0.1049	0.7777
7	1.1322	0.1322	0.9099
8	3.0901	0.0901	1.0000

Table (3)

* Let the random no. $r_1 = 0.763355$. We now select solution 6'

* We can see that the 1st part of stochastic remainder roulette-wheel selection operator is deterministic because the number of copies is decided using the integer value.

* It is considered less noisy than fitness proportionate selection operator in the sense of introducing less variance in its realization

2.3 Stochastic Universal Sampling (SUS):

— Refer Table (1)

* only one random number is required for selecting solutions
* since N different solutions have to be chosen a set, a set of N equi-spaced numbers is created
* $R = \{ r, r + 1/N, r + 2/N \ldots\ldots r + (N-1)/N \}$

Let $r = 0.40416$ and $N = 8$ $\Rightarrow 1/N = 0.125$

$\Rightarrow$ soln 3 is selected

$\Rightarrow r + 1/N = 0.0416 + 0.125 = 0.52916 \Rightarrow$ soln 7 is selected

* The series of numbers for selecting solutions is 0.65416, 0.77916, 0.90416, 0.02916, 0.15416 ...
* The following soln are then selected '8', '8', '8', '2', '3',

Limitations of FPS:

* Domination of "super solution" in early generations.
  * Suppose the fitness (maximization) of five soln in the population is given as

$f_1 = 10$, $f_2 = 5$, $f_3 = 70$, $f_4 = 7$ & $f_5 = 8$

  * Because of the fitness proportinate selection, solution '3' will get more copies and will become a "super-solution".

* Slower convergence in later generations:-

Suppose the fitness of five solutions in the population is given as

$f_1 = 19,\ f_2 = 21,\ f_3 = 22,\ f_4 = 18,\ f_5 = 20$

* Every sol^n will get a copy. It means that selection operator has no role in selecting solutions and thus, it becomes dummy.

Possible remedies:-

* Fitness scaling
* use ranking selection to avoid fitness scaling
* Tournament selection: it does not have any scaling problem.

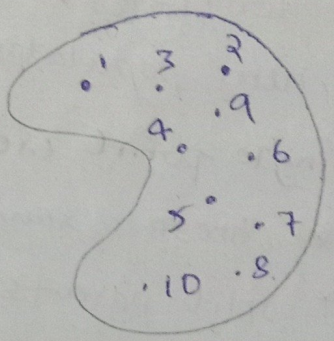
1. Tournament Selection operator:-

Tournament selection with tournament size k:

Randomly, we sample a subset $\hat{P}$ of k solutions from the population P. select solution in $\hat{P}$ with best fitness.

* Suppose $k=4$, we choose four sol^ns randomly, say $\{2, 3, 6, 9\}$ and their fitness values are

$f_2 = 10,\ f_6 = 5.9,\ f_3 = 16.4,\ f_9 = 9.4$



⇒ For max? ⇒ solution is 3

for min ⇒ solution 6

* often, tournament size $k=2$ is used, which is known as binary tournament selection operators.

Operators for survivor or Elimination.

1. $(\mu + \lambda)$ selection scheme:
   * $\mu \rightarrow$ size of parent population, λ number of offspring solutions are generated after variation operator.
   * The next population is filled by choosing the μ best number of solutions from the combined populations of parent and offspring solutions.

2. (μ, λ) selection scheme where $\lambda > \mu$.
   * Note that $\lambda > \mu$. The next population is filled by choosing the μ best number of solutions from the offspring population only.

Crossover Operators for Binary Variables:

* Cross over operator create offsprings/new solutions from more than one parent
* Crossover probability (P_c)
  * $100\,P_c\%$ strings are used in the crossover operator
  * $100(1-P_c)\%$ of the population are simply copied to the new population.
* Single point cross-over operator:
  * choose a random site on the two parents
  * Split parents at this crossover site.
  * Create children by exchanging tails.
* n-point crossover:
  * choose n random crossover sities
  * Split along those sities
  * Glue parts, alternating between parts.

* 2-point crossover operator:

parent 1: 101|011|1010

parent 2: 010|100|1110

O-F 1: 101|100|1010

O-F 2: 010|011|1110

↳ changed.

* Uniform crossover:
  * Select a bit-string z of length n uniformly at random
  * for all i in 1 to n
    * if $z_i = 1$ then bit i in offspring-1 is x_i else y_i
    * if $z_i = 1$ then bit i in offspring-2 is y_i else x_i

parent 1: 1010111010

parent 2: 0101001110

---

z = 1010001110

of-1: 1111001010

of-2: 0000111110

Mutation Operator for Binary String:

The mutation operator introduces small, random changes to a solution's chromosome.

Local Mutation:

* one randomly chosen bit is flipped
* 101 0 1110 10
  101 1 1110 10

Global Mutation:

* Each bit flipped independtly with a given probability Pm.

* It is also called the per bit mutation rate, which is often $= 1/n$, where n is the chromosome length.

$P_m = 1/n$

* 1 0 1 0 1 1 1 0 1 0
  1 0 0 0 1 1 1 1 1 0

* $P_r[k \text{ bits flipped}] = \binom{n}{k} \cdot p_m^k \cdot (1-P_m)^{n-k}$
  └ Not understood.

Observation:-

* On increasing the string length, BGA giving good results.
* But still on increasing the string length (l) [means the precision] the algo can't reach to optimum value.

For high accuracy problems

* Even though Himmelblau fn (is 'multimodal' it has 4 minimums) BGA found all four optimum points.

Issues with Binary-Coded Genetic Algorithm:-

* Binary GA makes the search space discrete
* Hamming cliffs: The decoded values of string (10000) and string (01111) are 16 & 15. However, each bit of these string is different
* Arbitrary precision:

$$((x_i^{(U)} - x_i^{(L)}))/2^l - 1$$

impossible due to fixed-length coding.

Remedy:

* We should code decision variables as real number directly.

* However, crossover and mutation operators need structural changes.
* Important to note that selection operator remains the same because it requires fitness value.

LEC 5: Real coded GA

Algorithm:- (Similar to BGA, only changes are below)

1. Solution representation % real number

Q(t) := variation(M(t)) % Crossover & Mutation

Real coded EC Techniques:

* Real-coded genetic algorithm (RGA)
  * Similar to binary-coded GA
* Evolutionary strategies (ES)
  * one of the oldest EC techniques
* Differential evolution (DE)
* Particle swarm optimization (PSO)
  * Motivated from flocking of birds
* Artificial bee colony (ABC), etc
  * Motivated from foraging behavior of swarm of bee

problem: Rosenbrock function

* It is very similar to GBA, but we take near real values.

→ selection is same.

Crossover :-

* Crossover is performed with probability (P_c). Generally the value of P_c is kept high that supports exploration of search space.

old index	new index	x_1	x_2	$f(x_1, x_2)$
7	1	-0.742	1.934	194.618
4	2	-0.639	1.694	167.414
3	3	-2.393	-4.790	1066.8
1	4	2.212	3.009	357.15
1	5	2.212	3.009	357.15
2	6	-2.189	-2.396	583.51
7	7	-0.742	1.934	194.61
4	8	-0.639	1.692	167.41

Single-Point Cross-over in BGA :

Property-1 : The average of decoded values of binary strings before and after crossover are the same

parents	offsprings
10|1001	10|1010
01|1010	01|1001

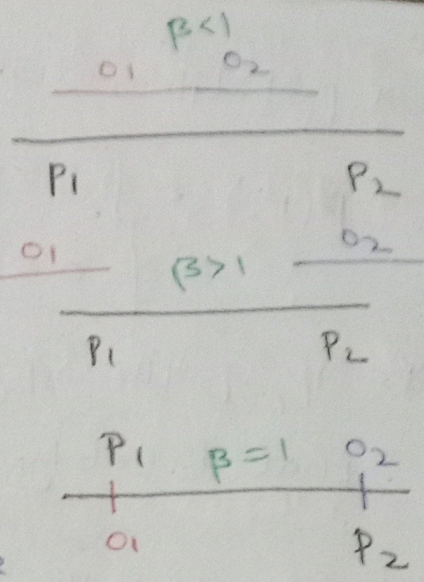
* Decoded values are : 41 & 26 & DV's are = 42 & 25

Avg = 33.5 Avg = 33.5

Property-2 : Spread factor β is defined as the ratio of the spread of offsprings solutions to that of parent solutions.

$$\beta = \left| \frac{O_2 - O_1}{P_2 - P_1} \right| \quad -(1)$$

* Contracting crossover, $\beta < 1$
  * The offspring solutions are enclosed by parent solutions.
* Expanding crossover, $\beta > 1$
  * The offspring solutions enclose the parent solutions.
* stationary crossover, $\beta = 1$
  * The offspring solutions are the same as parent solutions.



* We need structural change in crossover operator for dealing with real parameters.
* Simulated binary crossover (SBX) crossover operator
  : It was designed with respect to one-point crossover property in binary coded GA.
  * Average property :- property 1
  * Spread factor property :- property 2
* Now offsprings are calculated using average property

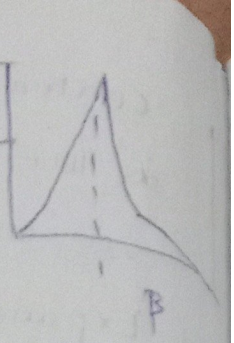
offspring: $o_1 = \bar{x} - \frac{1}{2}\beta(p_2 - p_1)$
$o_2 = \bar{x} + \frac{1}{2}\beta(p_2 - p_1)$
where $\bar{x} = \frac{1}{2}(p_1 + p_2)$

Δ $p_2 > p_1$

* It ensures that the average value of offspring and parent solutions are $\bar{o} = \bar{p}$
* probability distribution of β in SBX should be similar to the probability distribution of β in BGA.

* probability distribution function:

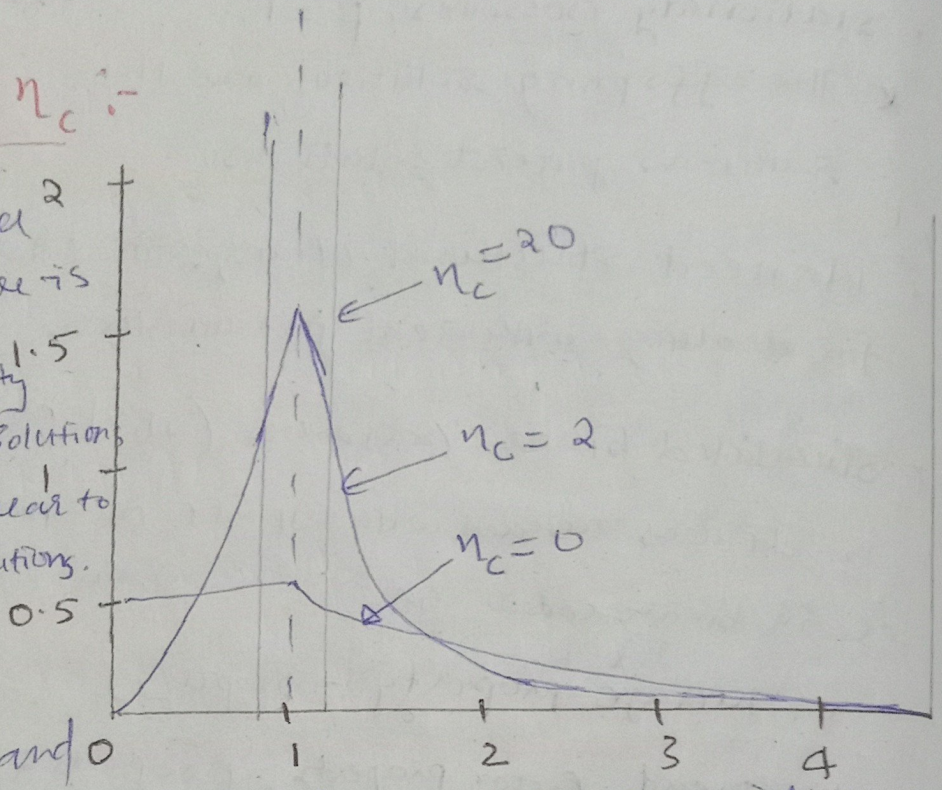
$$P(\beta_i) = \begin{cases} 0.5(\eta_c+1)\beta_i^{\eta_c}, & \text{if } \beta_i \le 1 \\ 0.5(\eta_c+1)\frac{1}{\beta_i^{\eta_c+2}}, & \text{otherwise.} \end{cases}$$



* η_c is the SBX crossoperator distribution factor that is set by us. (user defined)

Effect of η_c :-

* For a larger value of η_c, there is higher probability that offspring solutions will be created near to their parent solutions.



* Contracting and expanding on the sides of $\beta = 1$ value for different values of η_c

* calculate β_i by equating area under the probability curve equal to u_i (a random number $\in [0,1]$)

$$\beta_i = \begin{cases} (2u_i)^{\frac{1}{\eta_c+1}}, & \text{if } u_i \le 0.5 \\ \left(\frac{1}{2(1-u_i)}\right)^{\frac{1}{\eta_c+1}}, & \text{otherwise} \end{cases}$$

* offsprings are:

$$o_i^{(1,t+1)} = 0.5\left[(p_i^{(1,t)} + p_i^{(2,t)}) - \beta_i\left(p_i^{(2,t)} - p_i^{(1,t)}\right)\right]$$

$$o_i^{(2,t+1)} = 0.5\left[(p_i^{(1,t)} + p_i^{(2,t)}) + \beta_i\left(p_i^{(2,t)} - p_i^{(1,t)}\right)\right]$$

* Here, $\beta_i^{(1,t+1)}$ represents i-th decision variable of the first parent solution at (t+1)-th generation. Also $P_1^{(1,t)} < P_i^{(2,t)}$.

$[\because P_1 < P_2]$

SBX crossover operator: Hand calculations:

Mating pool

Row Index	x_1	x_2	$f(x_1, x_2)$
1	-0.742	1.934	194.618
2	-0.639	1.692	167.414
3	-2.393	-4.790	11066.800
4	2.212	+3009	357.154
5	2.212	3.009	557.154
6	-2.259	-2.396	5943.569
7	-0.742	1.934	194.61
8	-0.639	1.692	167.414

Let us assume that the probability of crossover is $P_c = 0.9$ and user defined parameter of SBX cross operator is

$\eta_c = 15$

* pair	Random num (r)	pair	Random number (r)
{3,7}	0.63	{8,4}	0.98
{5,2}	0.13	{6,1}	0.57

offspring sol^n :-

$$x_i^{(1,t+1)} = 0.5\left[\left(x_i^{(1,t)} + x_i^{(2,t)}\right) - \beta_i\left(x_i^{(2,t)} - x_i^{(1,t)}\right)\right]$$

$$x_i^{(2,t+1)} = 0.5\left[\left(x_i^{(1,t)} + x_i^{(2,t)}\right) + \beta_i\left(x_i^{(2,t)} - x_i^{(1,t)}\right)\right]$$

also written as

$$= 0.5\left[(1+\beta_i)x_i^{(1,t)} + (1-\beta_i)x_i^{(2,t)}\right]$$

$$= 0.5\left[(1-\beta_i)x_i^{(1,t)} + (1+\beta_i)x_i^{(2,t)}\right]$$

* For the first pair {3,7}, since $r = 0.63 < P_c = 0.9$, we perform crossover.

* Let us perform on x_1 variable of both solutions, that is $x_1^{(3)} = -2.393$ and $x_1^{(7)} = -0.742$

* It is important to note that when we perform cross-over between two solutions P_1 and P_2 $[P_1 < P_2]$, otherwise interchange the values.

For this pair $P_1 = x_1^{(3)} = -2.393$ & $P_2 = x_1^{(7)} = -0.742$

Random number $u_1 = 0.236$, The value w.r.t u_1 is

$\beta_1 = 0.954$.

$\Rightarrow$ The new offsprings are $0.5[(1+\beta_1)P_1 + (1-\beta_1)P_2]$ and $0.5[(1-\beta_1)P_1 + (1+\beta_1)P_2]$, they are -2.355 & -0.78

* crossover on x_2 variable of both sol^{ns}, that is

$P_1 = x_2^{(3)} = -4.790$ & $x_2^{(7)} = -0.780$ $1.934 = P_2$

on substituting in formula we get new offsprings as -4.773 & 1.917

* The new soln are: $(-2.355, -4.773)^T$ & $(0.780, 1.917)^T$

11^{th} on doing for pair {5,2}, we get we sol^n as $(-0.785, 1.691)^T$ & $(2.359, 5.010)^T$

11^{th} we done for others and got the solutions after crossover

Solutions after crossover

Index	x_1	x_2	$f(x_1, x_2)$	
1	-0.809	1.881	153.894	} the best fitness among soln before crossover was 167.414
2	-0.785	1.772	125.261	
3	-2.355	-4.973	⋮	
4	2.122	3.009		
5	2.359	2.978		
6	-2.223	-2.342		
7	-0.780	1.417		
8	-0.639	1.692		

* Better solutions will be emphasized and bad solutions will be eliminated by selection operator in further generations.

Mutation operator:

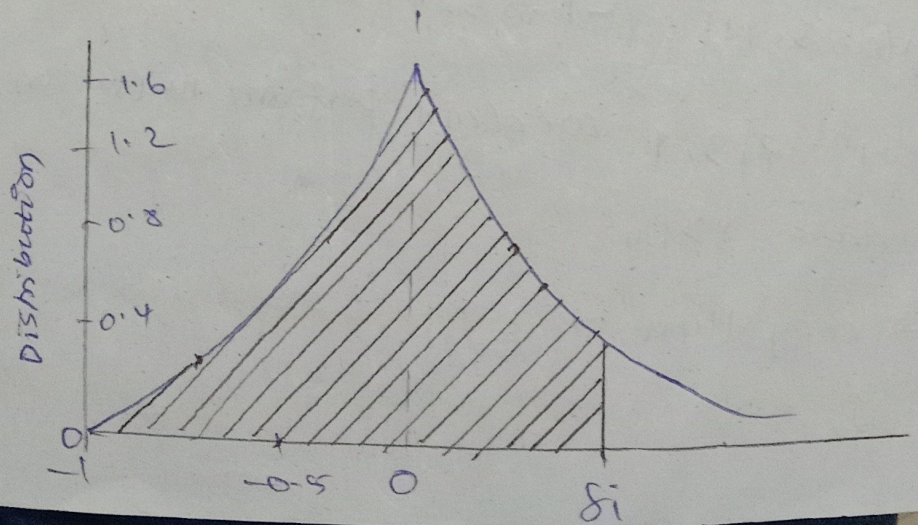
purpose: create new solution in a population with a low probability (p_m): exploitation.

polynomial mutation operator:

* $y_i^{(1,t+1)} = x_i^{(1,t+1)} + (x_i^{(U)} - x_i^{(L)})\,\bar{\delta}_i$

* $\bar{\delta}_i$ is calculated from polynomial probability distribution

$$P(\delta) = 0.5\,(\eta_m + 1)\,(1 - |\delta|)^{\eta_m}$$



* Calculate $\bar{\delta}_i$ by equating area under the probability curve equal to r_i (a random number $\in [0,1]$)

$$\bar{\delta}_i = \begin{cases} (2r_i)^{1/(\eta_m+1)} - 1 & \text{if } r_i \leq 0.5 \\ 1 - [2(1-r_i)]^{1/\eta_m+1} & \text{if } r_i \geq 0.5 \end{cases}$$

η_m is user defined.

* $P_m = 1/n = 0.5$ ($n=2$ ∵ two variables x_1 & x_2) &

$\eta_m = 20$

* we create random number between [0 1] for each solution to perform mutation. Let the random numbers are: 0.24, 0.62, 0.98, 0.79, 0.31, 0.71, 0.49 & 0.83

For soln 1:

$r_1 = 0.956$ for $x_1^{(1)} = -0.809$, we calculate

$\bar{\delta}_1 = 0.109$ (using above formulas)

$\Rightarrow y_1^{(1)} = x_1^{(1)} + (x_1^{(0)} - x_1^{(L)})\,\bar{\delta}_1$ → How ?

$= -0.809 + (10)(0.109) = 0.284$

* For x_2 Let $r_2 = 0.635$ then $\bar{\delta}_2 = -0.003$

then $y_2^{(1)} = 1.831 + (10)(-0.03) = 1.858$

$\Rightarrow$ New soln : $[-1.283, 1.856]^T$

* For soln 2, 3, 4 we don't perform mutation because $r > P_m$

→ copy them.

11.4 done for other solutions we get

Solutions after Mutation

Index	x_1	x_2	$f(x_1, x_2)$
1	0.254	1.856	315.568
	-0.785	1.772	1252
	-2.355	-4.773	10655.9
	2.212	3.009	357.15
	1.969	3.104	60.7
	-2.223	-2.342	5313.910
	-0.528	0.564	10.10
	-0.639	1.692	167.414

* Best soln in intial population: $(-0.639, 1.692)^T$ with $f(x_1, x_2) = 167.414$.

→ Best soln after crossover $(-0.785, 1.691)^T$ with 11847

→ Best soln after mutation: $(-0.528, 0.564)^T$ with 10.515

* However 1 soln got woorse after mutation

Survivor or Elimination:

* $(\mu + \lambda)$ - strategy

parent population

	x_1	x_2	$f(x_1, x_2)$
1			357.15
2			5843.5
3			11066
* (4)			167 ← same
5			8718
6			574
(7)			194
8			2575)

offspring population

	x_1	x_2	$f(x_1, x_2)$
(1)			351
(2)			125
3			10655
4			357
(5)			60
6			5313
(7)			10.5
(8)			167

Next generation population

Var index	x_1	x_2	$f(x_1, x_2)$
1			10.515
2			60
3			125
4			167
5			167
6			194
7			315
8			357

* first generation is over, increase the counter by one i.e $t = t+1 = 2$. Check termination cond?

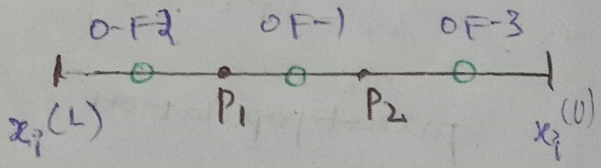
LEC-6: OPERATORS AND SIMULATION OF RGA.

* Crossover operators of real parameters are also called as blending operators

1. Linear crossover operator:

parent 1: $P_1 = x_i^{(1,t)}$

parent 2: $P_2 = x_i^{(2,t)}$



offspring 1: $0.5\left(x_i^{(1,t)} + x_i^{(2,t)}\right)$

offspring 2: $\left(1.5\, x_i^{(1,t)} - 0.5\, x_i^{(2,t)}\right)$

O-F3: $\left(-0.5\, x_i^{(1,t)} + 1.5\, x_i^{(2,t)}\right)$

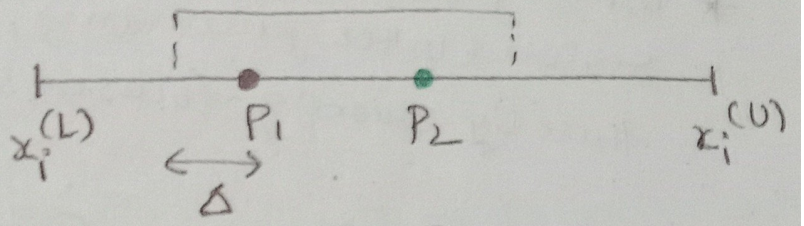
Here $x_i^{(1,t)}$ represents i-th decision variable of the randomly chosen first parent in the t-th generation

* we choose two best offspring solutions among three.

2. Blend crossover and its variants:

* Blend crossover (BLX-α). The assumption is $P_1 = x_i^{(1,t)} < x_i^{(2,t)} = P_2$ for two parents

* $\Delta = \alpha(P_2 - P_1) = \alpha(x_i^{(2,t)} - x_i^{(1,t)})$



* BLX-α randomly picks a solution in the range

$$\alpha \in [x_i^{(1,t)} - \Delta,\ x_i^{(2,t)} + \Delta]$$

* offspring: $\boxed{x_i^{(1,t+1)} = (1-\gamma_i)x_i^{(1,t)} + \gamma_i x_i^{(2,t)}}$, where

$\boxed{\gamma_i = (1+2\alpha)u_i - \alpha}$ → user defined

u_i - randomly generated by computer $\in \{0,1\}$

* It can be seen from the figure that γ_i is uniformly distributed for a fixed value of α.

* The value of $\boxed{\alpha = 0.5}$ is found to be performing better than other α values (Deb, 2001) over many test problems.

* Let us rewrite the equation for BLX-α

$$\underbrace{x_i^{(1,t+1)}}_{OF} = (1-\gamma_i)x_i^{(1,t)} + \gamma_i x_i^{(2,t)}$$

$$\Rightarrow (x_i^{(1,t+1)} - x_i^{(1,t)}) = \gamma_i (x_i^{(2,t)} - x_i^{(1,t)})$$

* It suggests that the location of offspring depends on the difference in parent solutions.

* The above condition signifies an adaptive search.
* In intial generations when random population is distributed over the entire search space, the diffe is large and an offspring population with a large diversity is expected.
* when population is large tends to converge in sm region (later generations), difference is small thus by causing focused search.

3. Simulated Binary Crossover (SBX) operato

probability distribution function:

$$P(\beta_i) = \begin{cases} 0.5(\eta_c+1)\beta_i^{\eta_c} & \text{if } \beta_i \le 1 \\ 0.5(\eta_c+1)\frac{1}{\beta_i^{(\eta_c+2)}} & \text{otherwise} \end{cases}$$

→ calculate β_i by equating the area under the probability function equal to u_i (a random number $\in [0,1]$)

$$\beta_i = \begin{cases} (2u_i)^{\frac{1}{\eta_c+1}} & \text{if } u_i \le 0.5 \\ \left(\frac{1}{2(1-u_i)}\right)^{\frac{1}{\eta_c+1}} & \text{otherwise} \end{cases}$$

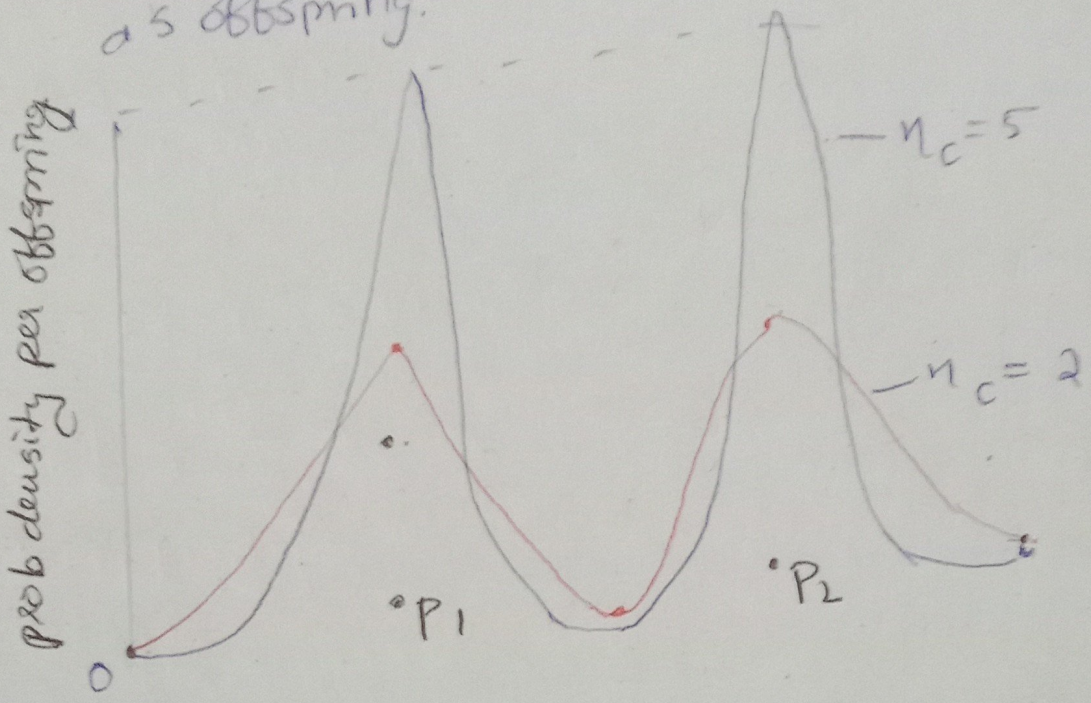
Offsprings are:

$$x_i^{(1,t+1)} = 0.5\left[(1+\beta_i)x_i^{(1,t)} + (1-\beta_i)x_i^{(2,t)}\right]$$

$$x_i^{(2,t+1)} = 0.5\left[(1-\beta_i)x_i^{(1,t)} + (1+\beta_i)x_i^{(2,t)}\right]$$

Remember: $x_i^{(1,t)} < x_i^{(2,t)}$ i.e $P_1 < P_2$

* Large value of η_c indicates higher prob of creating 'near-parent' offspring.
* small η_c allows distant solutions to be created as offspring.



properties of SBX crossover:

* The difference between the offspring is in proportion to the parent solutions.

$$\left(x_i^{(2,t+1)} - x_i^{(1,t+1)}\right) = \beta_i \left(x_i^{(2,t)} - x_i^{(1,t)}\right)$$

* Near parent solutions are monotonically more likely to be chosen as offspring than solutions distant from parent

LEC-8: Particle Swarm Optimization (PSO)

* PSO is developed using two methodologies
  * Artificial life: Mimicking bird flocking, fish schooling and swarming theory.
  * Evolutionary computation.
* The swarm searches for the food in a cooperative way.
* Each member in the swarm learns from its experience and also from other members for changing the search pattern to locate the food.
* PSO is computationally inexpensive both in memory and speed, and also can be easily implemented using computer programming.
* PSO starts with intializing population randomly similar to GA
* Unlike GA operators, solutions are assigned with randomized velocity to explore the search space.
* Each solution in PSO is referred to as particle

Three distinct features of PSO

Best fitness of each particle	Best fitness of swarm	Velocity and position update of each particle

* $pbest_i$: the best solution (fitness) achieved so far by particle 'i'.

* gbest: The best solution (fitness) achieved so far by any particle in the swarm
* Velocity and position update: For exploring and exploiting the search space to locate the optimal solution.

Position and Velocity:-

* position of particle (i) is adjusted as

$$x_i^{(t+1)} = x_i^{(t)} + v_i^{(t+1)}$$

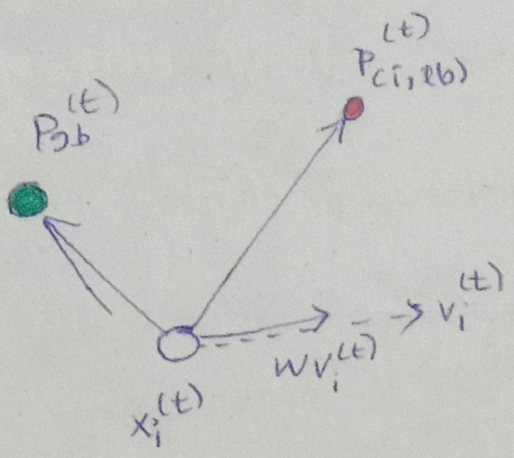
* Velocity of particle (i) is updated as follows:

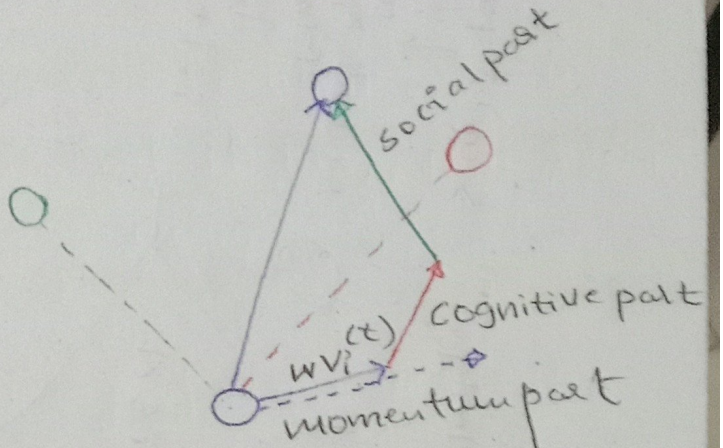
$$v_i^{(t+1)} = \underbrace{w v_i^{(t)}}_{\text{momentum}} + c_1 r_1 \left(P_{(i,lb)}^{(t)} - x_i^{(t)}\right) + c_2 r_2 \left(P_{gb}^{(t)} - x_i^{(t)}\right)$$

- i — ith particle
- t — generation counter
- $v_i^{(0)}$ — set randomly
- w — adds to the inertia of the particle
- c_1 & c_2 are the acceleration coefficients
- r_1 and r_2 are random numbers $\in [0\ 1]$
- $P_{(i,lb)}^{(t)}$ is the local best of i^{th} particle
- $P_{gb}^{(t)}$ is the global best

* Momentum part: $wv_i(t)$
  * Inertia component.
  * memory of previous flight direction.
  * prevents particle from drastically changing direction
* Cognitive part: $c_1 r_1 \left(P_{(i,lb)}^{(t)} - x_i^{(t)}\right)$
  * quantifies performance relative to past performances
  * Memory of previous best position
  * nostalgia.

* Social part: $c_2 r_2 \left(p_{gb}^{(t)} - x_i^{(t)} \right)$
* quantifies performance relative to neighbors.
* envy.





Geometrical illustration

Local & Global best positions

* $p_{(i,lb)}^{(t)}$ is the personal best or local best of ith particle in t-generation. Assume minimization problem

$$p_{(i,lb)}^{(t+1)} = \begin{cases} x_i^{(t+1)} & \text{if } f(x_i^{(t+1)}) < f(p_{(i,lb)}^{(t)}) \\ p_{(i,lb)}^{(t)} & \text{otherwise} \end{cases}$$

* $p_{gb}^{(t)}$ is the global best-position in t-generation which is calculated as

$$p_{gb}^{(t)} \in \{ p_{(1,best)}^{(t)} \dots\dots p_{(N,best)}^{(t)} \}$$

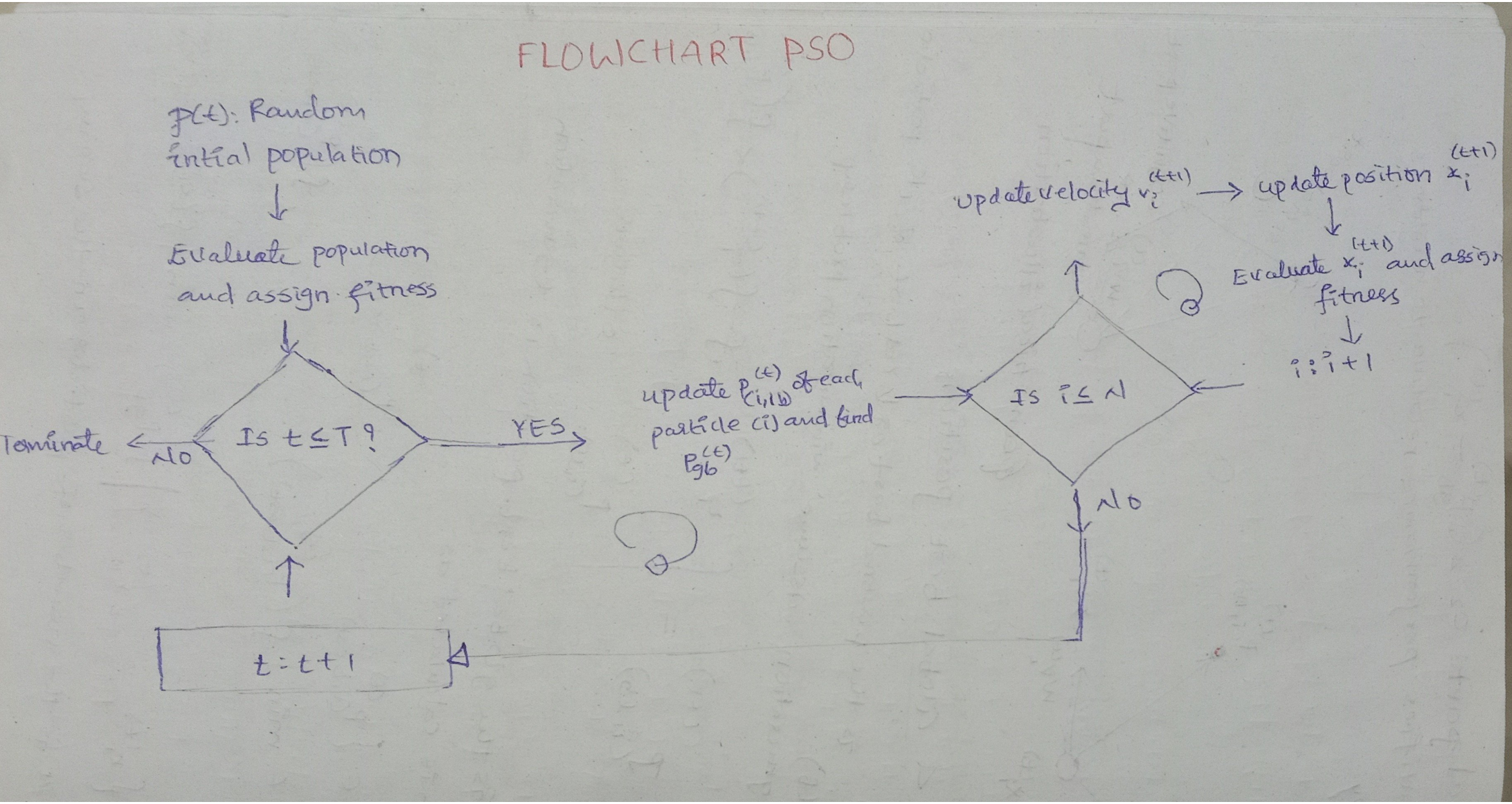
$$f(p_{gb}^{(t)}) \in \min \{ f(p_{(1,best)}^{(t)}) \dots\dots f(p_{(N,best)}^{(t)}) \}$$

(or)

$$p_{gb}^{(t)} \in \{ x_1^{(t)}, \dots\dots x_N^{(t)} \} \Big| f(p_{gb}^{(t)}) = \min \{ f(x_1^{(t)}) \dots f(x_N^{(t)}) \}$$

where N is the number of particles in the swarm

FLOWCHART PSO



Algorithm [Generalized framework]

1. Solution representation
2. Input: $t := 1$, T (max)
3. Initialize random swarm ($p(t)$); % swarm
4. Evaluate ($p(t)$); % Evaluate objective, constraints and assign fit
5. while $t \leq T$ do
   update $p^{(t)}_{(i,gb)}$ of each particle (i) and find $p^{(t)}_{gb}$;
   for ($i = 1$; $i \leq N$; i++) do
   update velocity ($v_i^{(t+1)}$);
   update position ($x_i^{(t+1)}$);
   Evaluate ($x_i^{(t+1)}$) and include it in $p(t+1)$;
   End for
   $t = t + 1$
   end while