Unit 4

Soft Computing

Genetic Algorithms

* Evolutionary Computation

→ Computational techniques inspired by biological evolution

→ Mostly has meta-heuristic optimization algorithms such as:

(i) evolutionary algorithms - genetic algorithms

(ii) swarm intelligence - ant colony optimization & particle swarm optimization

* Advantages of Evolutionary Computation

→ simplicity

→ robust to dynamic changes

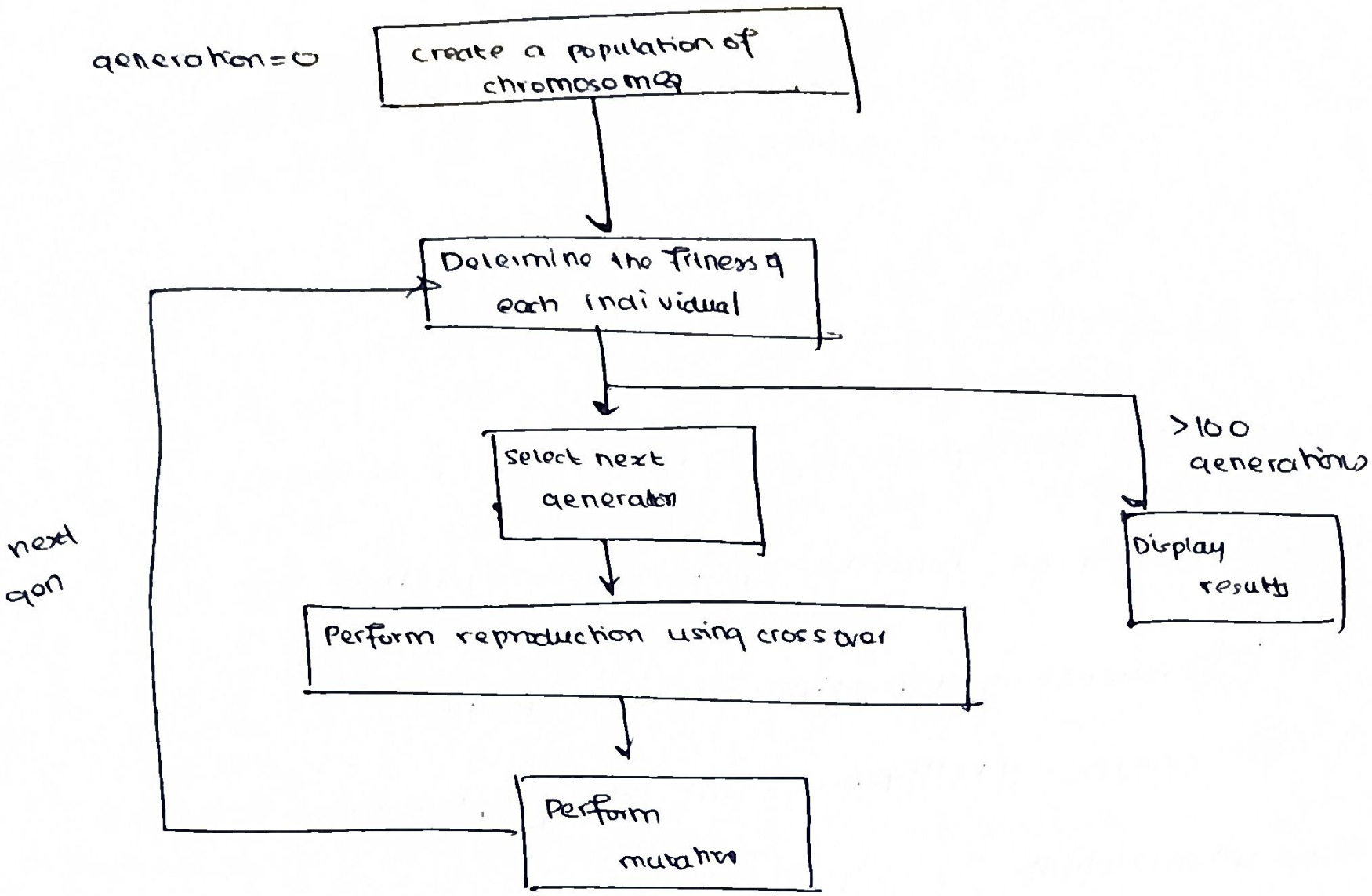
→ parallelism

* Genetic Algorithms

→ inspired by Darwin's Theory of natural evolution

→ search the space of individuals for good candidates

→ chance of an individual being selected is proportional to the amount by which its fitness is greater or less than its competitor's fitness.



Steps:

1. [Start] - Generate a random population of n chromosomes

2. [Fitness] - Evaluate the fitness f(x) of each chromosome x in the population

3. Repeat until the termination criterion is satisfied:

   (i) [Selection] - select 2 parent chromosomes from a population according to their fitness

   (ii) [Crossover] - crossover parents to form offspring

   (iii) [Mutation] - mutate offspring at selected positions in the chromosome

   (iv) [Accepting] - Generate new population by placing new offspring.

* Loop Termination Methods

(i) reaching some known / hoped for fitness

(ii) reaching a maximum allowed no. of generations

(iii) reaching some minimum level of diversity

(iv) reaching some specified no. of generations without any improvement.

* Criteria for GA Approaches

(1) Completeness – any solution should have its encoding

(2) Non-redundancy – codes and solutions should correspond 1-1

(3) Soundness – any code should have its solution

(4) Characteristic preservation – offspring should inherit useful characteristics from parents

* Genetic Algorithm Components

(1) Fitness function

→ represents the requirements the population should adapt to

→ quality / objective function

→ calculate by decoding chromosome & then evaluating the objective function

→ an indicator of how close the chromosome is to the optimal solution

→ typically: maximize fitness

(2) Selection

→ depends on fitness - assign probabilities

→ high quality solutions more likely to become parents

→ the stochastic nature can aid escape from local optima

Survivor Selection + Replacement

→ fitness-based —(i)

→ age-based —(ii)

→ or combination of (i) & (ii)

(3) Chromosome Encoding

→ encode as binary string - each bit represents a characteristic

(4) Crossover

→ operate on selected genes from parents to create offspring

→ usually chosen random - choose pt 2 swap

Mutation

→ randomly change the offspring resulted from crossover

→ intended to prevent falling of all solutions

→ randomly switch 0 → 1 bits

* Crossover and Mutation Schemes

A. When in binary encoding

Crossover

(i) single crossover point

(ii) 2 point crossover : 11 0010 01 + 11 0111 11

= 11 0111 01

(iii) Arithmetic crossover : do AND

(iv) Uniform crossover: randomly copy bits from both parents

Mutation

(i) Bit Inversion — selected bits are inverted

B. When in permutation encoding → representation as numbers

Crossover : (i) single point crossover.

Mutation : (i) order changing

C. Value Encoding

* crossover - all from binary encoding

2. mutation : adding - add / sub a small no. to selected values

D. Tree Encoding

crossover - select one crossover pt

exchange parts below

mutation - changing operator, number

* GA Advantages & Disadvantages

* Genetic Programming?

→ starts with randomly generated computer programs & evolves programs progressively over a series of generations

→ find solns. where variables are constantly changing

→ construct a population of random trees

→ Genetic operators - crossover & reproduction are done on trees

Steps

(i) Define 2 sets

→ terminal set T ⇒ variables, constants

→ function set f

(ii) Generate a population of random compositions of the 2 terminals.

(iii) Execute each program - assign a fitness value

(iv) Create a new population of programs

(v) Copy the best ones

(vi) Perform crossover & mutation

(vii) Keep generating random trees until all the branches end in terminals

* Genetic Programming Coding Scheme

→ Every chromosome is a tree of objects like functions or objects

→ Use the LISP programming lang (LISP)

Examples → can also do mutation (randomly replace variables operators)

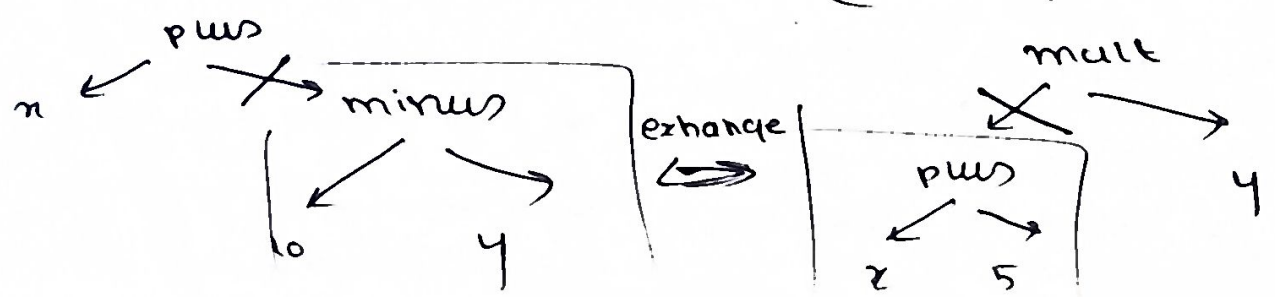
① Crossover

Chromosome A

x + (10 - 4) ⇒ + x (minus 10 4)

Chromosome B

(x + 5) * 4 ⇒ (mult (plus x 5) 4)



* Simulated Annealing, Random Search & Downhill Simplex Search

(1) Simulated Annealing

→ find near optimal solutions for large scale combinatorial optimization problems like the TSP / placement problem.

→ based on the analogy of what happens when metals are cooled at a controlled rate.

slow, steadily falling temperature ⇒ regular crystalline structure

cooling too fast ⇒ amorphous and irregular.

→ At high temperatures - SA allows function evaluation at faraway points, and is likely to accept a new point with higher energy.

→ At low temperates, SA evaluates the objective function only at local points.

→ The most important part of SA is the annealing schedule, which specifies how rapidly the temperature is lowered.

* Terminology

A. Objective fn: $E = f(x)$ ↳ each point in the input space

sample the i/p space to find an x that minimizes E

B. Generating Function: $q(\cdot,\cdot)$ specifies the PDF of the diff. between the current x next point to be generated.

$\Delta x = x_{new} - x$ = a random variable, has a PDF of

$q(\Delta x, T)$

C. Acceptance Function: After a new point x_{new} has been evaluated, SA decides whether to accept/reject.

Most frequently used acceptance fn is the Boltzmann probability distribution

$$h(\Delta, E, T) = \frac{1}{1 + \exp(\Delta E / (cT))}$$

$\Delta E = f(x_{new}) - f(x)$

D. Annealing Schedule - regulates how rapidly the temperature T goes from high to low values, as a function of time or iteration counts.

* Steps in General SA

(1) Choose a start point x and a high starting temperature T. Set the iteration count k to 1.

(2) Evaluate the objective function: $E = f(x)$

(3) Select Δx with a probability determined by the generating fn $q(\Delta x, T)$.

(4) Calculate the value of the new objective function:

$$E_{new} = f(x_{new})$$

(5) Set x to x_{new} and E to E_{new} with probability determined by the acceptance function $h(\Delta E, T)$ where $\Delta E = E_{new} - E$.

(6) Reduce the temperature T according to the annealing schedule i.e set $T = \eta T$, where η is between 0 and 1.

(7) Increment the iteration count k. If k reaches the maximum iteration count, stop the iterations. Otherwise, go back to step 3.

* Need for a Move Set

→ Each x in simulated annealing is confined to be one of N points that constitute the solution space / input space.

→ Usually N is finite, but it is also large enough that the use of an exhaustive search method is impossible.

→ Adding randomly generated Δx to a current point may not generate another legal point in the solution space, so generating functions are rarely used.

→ Instead, to find the next legal move to explore, use a move set, denoted by $M(x)$, as a set of legal points available for exploration after x.

→ The move set $M(x)$ represents a set of neighboring points of the current point x.

→ Once the move set is defined, x_{new} is usually selected at random from the move set, where every member stands an equal probability of being chosen.

* Solution to the Travelling Salesman Problem (TSP) using move sets

→ In a TSP, n cities are given, and the distance (cost) between all pairs of these cities, is given by a nxn matrix D, where

dij = distance / cost of traveling from city i to city j.

→ The problem is to find a closed tour in which each city, except for the starting one, is visited exactly once - such that the total length (cost) is minimized.

→ TSP is np-complete in nature, in which the computation time required to find an optimal solution increases exponentially with n.

→ With n cities : the number of possible tours is $\frac{(n-1)!}{2}$.

→ 3 different move sets can be defined to solve TSP using simulated annealing:

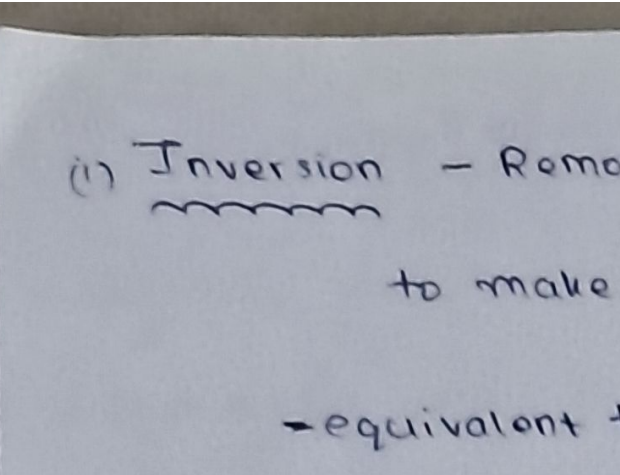
(i) Inversion – Remove two edges from the tour and replace them to make it another legal tour.

- equivalent to removing a section 6-7-8-9, and then replacing it with the same cities running in the opposite order.

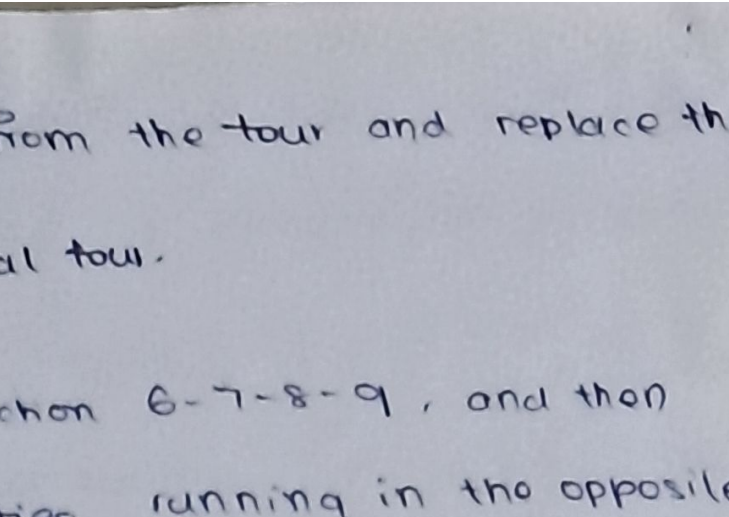
(ii) Translation – remove a section (8-7), and then replace it ~~with~~ in between two randomly selected consecutive cities

(iii) Switching – randomly select two cities (3 and 11) and switch them in a tour.

Example: Consider the following tour.



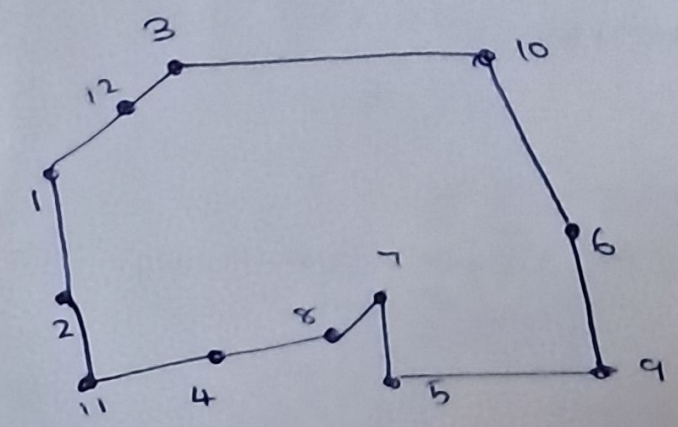
inversion of 6-7-8-9 ⟹



1-2-3-4-5-[6-7-8-9]-10-11-12

1-2-3-4-5-[9-8-7-6]-10-11-12

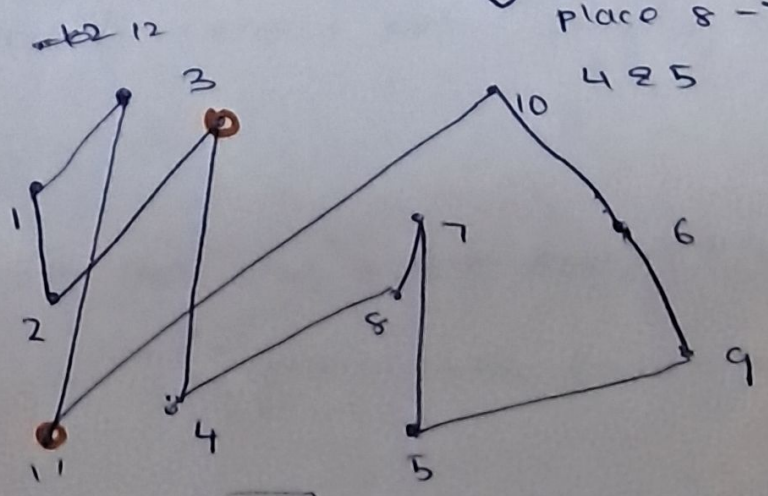
switching of 3211 ⟸



8-7

1-2-(11)-4-[8-7]-5-9-6-10-(3)-12

⟸ translation place 8-7 between 4 & 5



1-2-3-4-[5-7]-5-9-6-10-11-12

* Random Search

→ Random search explores the parameter space of an objective functions sequentially in a seeminaly random fashion to find the ophimal point that minimizes or maximizes the objective functions

→ $f(x)$: objective fn. to be minimized

x : point currently under consideration

→ The random search method trios to find the ophimal x by iterating the following 4 steps:

(i) Choose a start point x as the current point

(ii) Add a random vector dx to the current point x in the parameter space and evaluate the objective function at the new point $x + dx$.

(iii) If $f(x+dx) < f(x)$, set the current point x equal to $x + dx$.

(iv) Stop if the maximum number of function evaluations is reached. Otherwise, go to step 2.

Note that search directions are purely guided by a random number generated

Observations

(i) If search in a particular direction results in a higher objective function, the opposite direction can often lead to a lower objective function

(ii) Successive successful searches in a certain direction should bias subsequent searching in this direction

Modified Random Search with Bias

1. Choose a random start point x as the current point. Set the inital bias to a zero vector

2. Add a bias term b and a random vector dx to the current point x in the input space, and evaluate the objective function at the new point $x+b+dx$

3. If $f(x+b+dx) < f(x)$

$$x \leftarrow x+b+dx$$

$$b \leftarrow 0.2b + 0.4dx$$

Go to step 6

else go to the next step

4. If $f(x+b-dx) < f(x)$

$$x \leftarrow x+b-dx$$

$$b \leftarrow b - 0.4dx$$

Go to step 6

else go to the next step

5. Set $b \leftarrow 0.5b$ and go to step 6

6. Stop if the maximum number of function evaluations is reached. Otherwise go back to step 2 to find a new point.

* Downhill Simplex Search

→ It is a derivative - free method for multidimensional function optimization.

→ It aims to minimize a function of n variables with no constraints

→ Start with an initial simplex, which is a collection of n+1 points in an n-dimensional space.

→ The method starts with a simplex, which is a geometric shape that has one vertex more than the number of dimensions of the function being optimized.

For example, a simplex is a triangle in two-dimensional space and a tetrahedron in 3-dimensional space.

$$P_i = P_0 + \lambda_i e_i, \; i = 1 \ldots n$$

e_i = unit vectors constituting a basis of the n-dimensional space

λ_i = constant

y_i is written for the function value at P_i

$$y_l = \min_i(y_i)$$

$$y_h = \max_i(y_i)$$

There are 4 intervals in the search process:

Interval 1: $\{y \mid y \leq y_l\}$

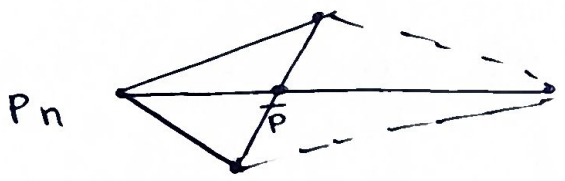
Interval 2: $\{y \mid y_l < y \leq \max_i \{y_i\}\}$

Interval 3: $\{y \mid \max_i \{y_i\} < y \leq y_h\}$

Interval 4: $\{y \mid y_h < y\}$

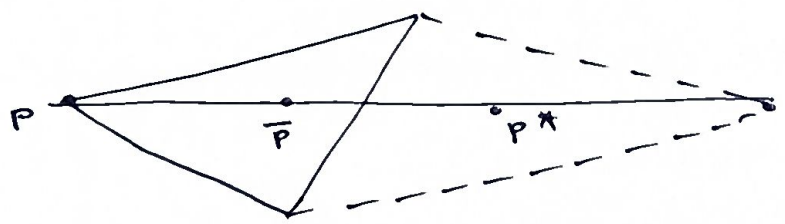
→ $\bar{P}$ is the average (centroid) of the n+1 points. Each cycle of this method starts with a reflection point P^* of P_h. Depending on the function value at P^*, there are 4 possible operations to change the simplex:

(i) reflection away from P_h



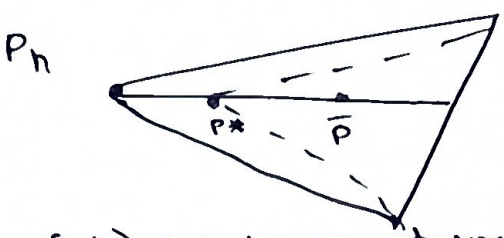
$$P^* = \bar{P} + \alpha(\bar{P} - P_n)$$

(ii) reflection and expansion away from P_h



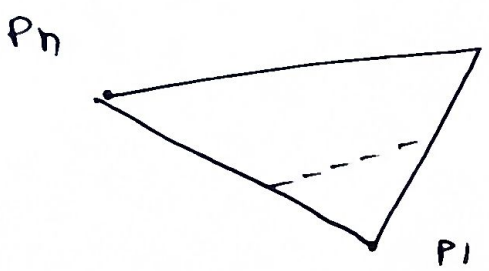
$$P^{**} = \bar{P} + \gamma(P^* - \bar{P})$$

(iii) contraction along one dimension connecting P_h and $\bar{P}$



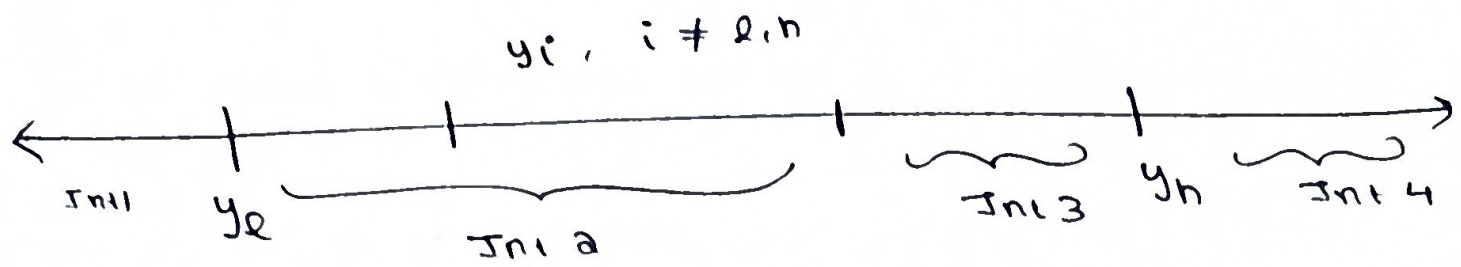
$$P^* = \bar{P} + \beta(P_h - \bar{P})$$

(iv) shrinkage towards P_l along all dimensions



$$P_i = \frac{(P_i + P_l)}{2}$$

The intervals in downnill simplex search are represented as follows:



If in Int1 → go to expansion

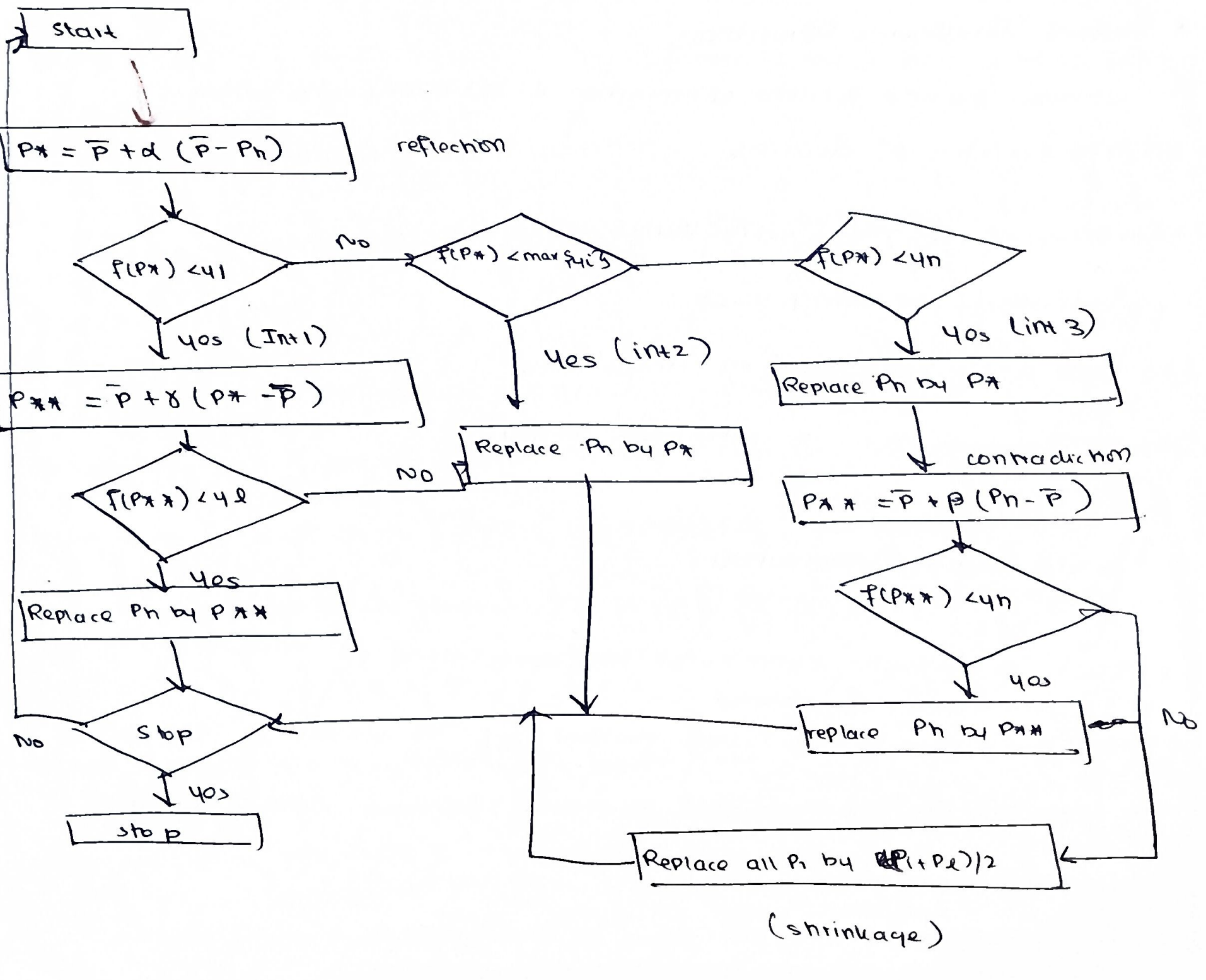
If in Int2 → replace Ph w/ P* and finish the cycle

If in Int 3 → replace Ph w/ P* and go to contraction

If in Int 4 → go to contraction

(X) (X)

Flow-Chart Representation



Drawbacks of Downhill Simplex Method

→ The method is deterministic in nature - starting from the same initial point, it will always lead to the same final point for a given objective function.

→ This could potentially lead to a local minimum and stay there forever.

→ To enable this method to get off of local minima, make it stochastic.

* Swarm Intelligence Algorithms

- include particle swarm optimization & ant colony optimization

* Characteristics of Swarms

(i) composed of many individuals

(ii) individuals are homogenous

(iii) local interaction based on simple rules

(iv) self- organization - no centralized control

* Particle Swarm Optimization

→ PSO is a stochastic optimization technique, based on movement & intelligence of swarms

→ A population-based search method w/ the position of the particle representing the solution and swarm of particles representing the searching agent.

→ PSO finds the minimum value for the function.

→ The idea is similar to bird flocks searching for food.

- Bird = a particle
- Food = solution

→ pbest represents the best solution (fitness) a particle has achieved so far

→ gbest represent the global best of all the particles within the swarm

→ The basic concept in PSO lies in accelerating each particle before its pbest and gbest locations, with a random weighted acceleration at each time.

* PSO Search Scheme and Algorithm

→ Particles constitute a swarm flying in the search space looking for the best solution.

→ Each particle is treated as a point (candidate solution) in an N-dimensional space, which adjusts its flying according to its own flying experience as well as the flying experience of other particles

→ Each particle tries to modify its position X as follows:

$$X(t+1) = X(t) + V(t+1)$$

$$V(t+1) = \omega V(t) + c_1 \times rand() \times (X_{pbest} - X(t)) + c_2 \times rand() \times (X_{gbest} - X(t))$$

where $V(t)$ = velocity at t
$X(t)$ = position
ω = inertia weight

c_1, c_2 = accelerating / learning factor
X_{pbest} = particle's best pos
X_{gbest} = global best pos

* PSO Algorithm

Input : randomly initialized position and velocity of particles

$X_i(0)$ and $V_i(0)$

Output : position of the approximate global minimum x^*

while terminatingcondition is not reached do:

for i=1 to number of particles do:

(i) calculate fitness function f

(ii) update personal & global best of each particle

(iii) update velocity of particle

(iv) update position of particle

end for

end while

Numericals

(i) GA - CAT2, maxIs

(ii) SA - CAT2

(iii) PSO ?

(iv) GA - CW Question

* Ant Colony Optimization

→ studies artificial systems that take inspiration from the behavior of real ant colonies, which are used to solve discrete optimization problems

(a population - based metaheuristic)

→ The optimization problem must be written in the form of path finding with a weighted graph

→ The idea of ACO is that ants find their food using the shortest path.

→ Ants deposit pheromones to remember the path. These pheromones evaporate w/ time.

→ Whenever an ant finds food, it marks it return journey with pheromones

→ Pheromones evaporate faster on longer paths.

→ The shorter paths serve as the way to food for most of the other ants.

→ The shortest path will be reinforced by the pheromones further, and finally the ants arrive at the shortest path.

Algorithm

Set parameters, initialise pheromone trails

while termination condition not met do

Construct Ant Solutions

Apply Local Search

Update Pheromone

end while

Unit 4 - Numericals

(1) Using particle swarm optimization, minimize the function

$$f(x) = x^3 - 4x^2 + 5x$$

The inital velocities and positions are as follows

x_1 : 4.2 v_1 : 0.3

x_2 : -3.4 v_2 : 0.2

x_3 : 1.9 v_3 : 0.1

Let c_1 = ~~0.5~~ 1.5 $w = 0.6$

$c_2 = 2.0$

Iteration 1

Step 1 : Calculate fitness

X_1 : $(4.2)^3 - 4(4.2)^2 + 5(4.2)$

74.088 - 70.56 + 2.1

X_2 :

X_3 :

Step2 calculate Pbest & gbest

If pnow < pbest ⇒ make it Pbist

make gbest the lowest of all values

Step3 recompute velocity & pos

$$V(t+1) = \text{~~crand~~}\ wV(t) + \{c_1 rand() \times (x_{pbest} - x(t))\} + \{c_2 rand() \times (x_{gbest} - x(t))\}$$

$$X(t+1) = X(t) + V(t+1)$$

Iteration 2 for each $X(t+1)$

calculate fitness again & repeat

(2) **GA-1** Apply GA to maximize the function $f(x) = x^2$ where x varies from 0 to 63. The initial population size is 4, which is provided as:

001100
110011
000101 and
100110

consider the crossover point as 4 for gen. the offspring from 001100, 110011 and 2 for 000101 & 100110.

Let the mutation chromosomes by 100 000, 00000 and 010010 and 001000. Compute fitness & choose the next chromosomes

Ans

Inital pop	x	f(x)	probability	expected count f(x)/avg.	actual count
001100	12	144	0.034	1.366	1
110011	51	2601	0.6172	2.47	2
000101	5	25	0.0063	0.023	0
100110	38	1444	0.3426	1.3706	1
	sum	4214			
	avg.	1053.5			

0 0 1 1 | 0 0 & 1 1 0 0 | 1 1

0 0 1 1 11 & 1 1 0 0 00

0 0 0 1 0 1 ⇒ 10 0 1 0 1

1 0 0 1 1 0 ⇒ 00 0 1 1 0

Mutation

① ~~0 0~~ 1 0 1 1 1 1 → 47

② 1 1 0 0 0 0 → 48

} next gen

③ 1 1 0 1 1 1 → 55

④ 0 0 1 1 1 0 → 14