LAB-08:

Parallel Cellular Algorithm:

Algorithm:

Function obj_fn (x):
    return x**2 - 4*x + 4

Function initialize_pop():
    create random grid with values in range [-10, 10], return grid.

Function evaluate_fitness (population):
    create empty array fitness
    for each cell in popln:
        fitness[cell] = obj_fn (pop[cell])
    return fitness

Function update_cell (pop, fitness, neigh_states):
    temp ← pop, neigh_radius - neigh_state
neighbourhood = {}
    for di from (-neigh_rad to neigh_rad)
        for dj from " :
            ni = i+di , nj = j+dj
            if (ni, nj) ≤ bounds:
                add (pop^n [ni nj])

    sort neighbourhood
    best_neigh = neighbour [0]

algorithm ():
    pop ← initialize_pop ()
    fitness ← eval_fitness (pop)
    best_sol ← None
    best_fit ← infinity
for each iteration from i to iterations:
    new_pop ← update_cell ()
    fitness ← evaluate ()
    find min (fitness)
    if min_fitness < best_fit:
        best_fit = min_fitness
        best_sol ← soln in new popln

Implementat

Image Process

→ Edge Detec

Identifies sign

boundaries of ob

Implementation:

Image Processing:

→ **Edge Detection**:

Identifies significant changes in pixel intensity that correspond to the boundaries of objects or features in an image.

Optimization via Gene Expression Algorithms:

Algorithm:

```
Function obj_fun (x, y):
    return x**2 + y**2
Function initialize_pop ():
    for i in size:
        pop[i] ← random value.

Function evaluate_fitness (pop):
    fitness ← []
    for ind in pop:
        fitness[ind] = obj_fun()
    return fitness.
Function select_parents (pop, fitness):
    prob ← 1/fitness - e
    probb ← sum(prob)
    indices ← random set of indices from population
    return pop [indices]
Function crossover (par1, par2):
    if random < crossover_rate:
        point ← random from 1 to num_
        child1 ← concatenate (parent1[0:point], parent2[0:point])
        child2 ← concatenate (parent2[0:point], parent1[0:point])
        return child1, child2
    else:
        return parent1, parent2.

Function mutate (individual):
    for i ← 0 to num_:
        if random < mut_rate:
            ind [i] = ind [i] + random (-1,1)
    return individual
```

Implementation:

Output:

Generation 1 : Best Fi
Generation 2 : Best Fi
Generation 3 : Best Fi
Generation 4 : Best Fi
Generation 5 : Best Fi

Differences b/n Genetic

Genetic Algo:

→ sol^ns are rep as strin
→ swapping parts of 2 ch

→ works well for opt
fixed structures & s
effort to maintain c

Implementation:

Output:

Generation 1 : Best Fitness = 1.9219
Generation 2 : Best Fitness = 3.8054
Generation 3 : Best Fitness = 0.4556
Generation 4 : Best Fitness = 0.6337
Generation 5 : Best Fitness = 0.9917

Differences b/w Genetic Algo & Gene Exp Algo:

Genetic Algo:	Gene Exp Algo:
→ sol^{n}s are rep as strings / binary dgts	→ rep as expression trees or linear chromosomes
→ swapping parts of 2 chromosomes	→ exchanging sub trees between chromosomes
→ works well for optimizing fixed structures & req. more effort to maintain diversity	→ more suited for problems where the sol^{n}s are not fixed & performs better compared to genetic algo.