VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB RECORD

Bio Inspired Systems (23CS5BSBIS)

Submitted by

Priyam Thakur (1BM23CS252)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
Aug-2025 to Jan-2025

B.M.S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "Bio Inspired Systems (23CS5BSBIS)" carried out by **Prerana P Jois (1BM23CS250),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements of the above mentioned subject and the work prescribed for the said degree.

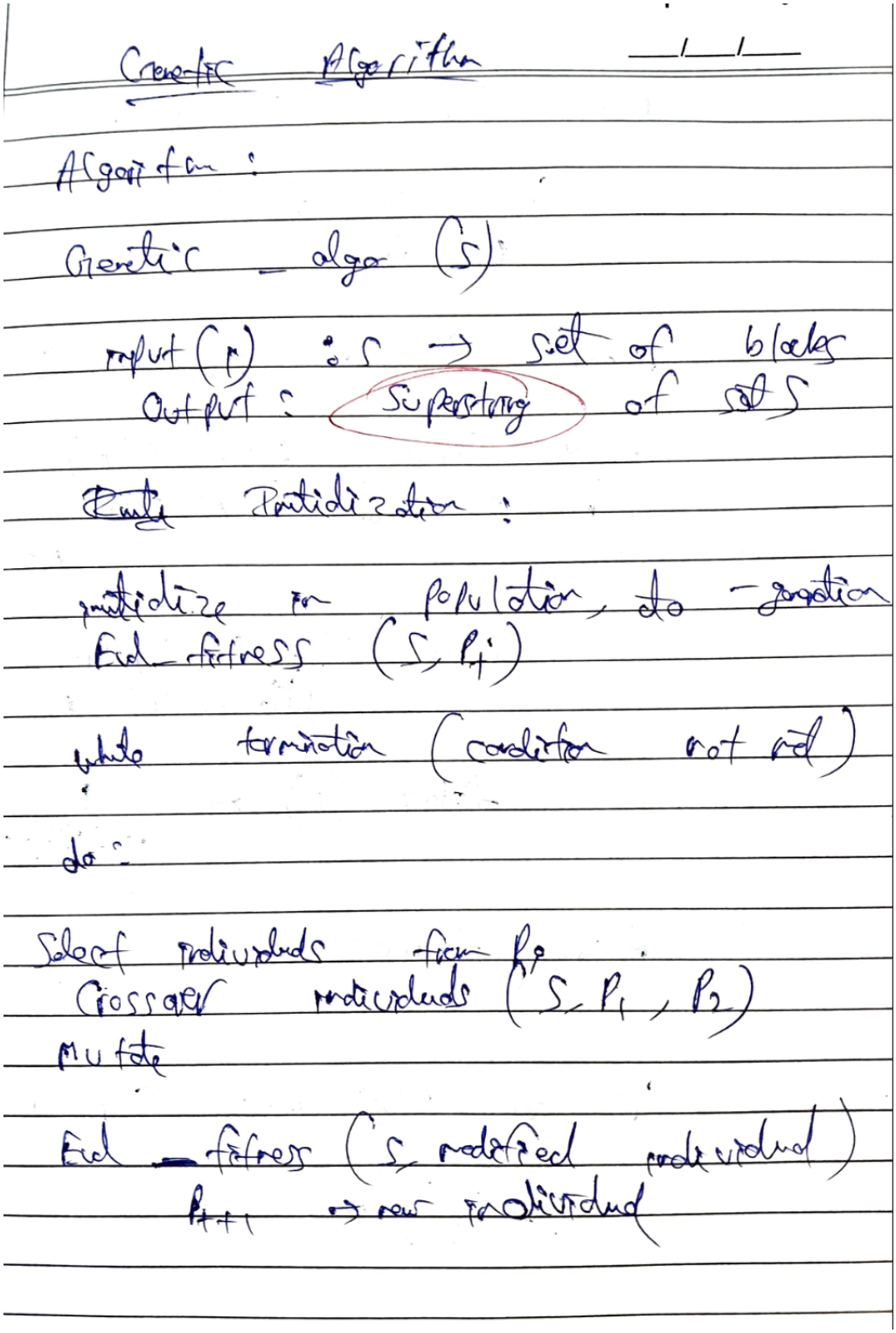
Swathi s Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

Index

Github Link:

https://github.com/priyamthakur275/BIS__LAB.git

Program 1:
Genetic Algorithm



find_fitness (S, Population):

S -> set of blocks
Population

for i in Population:

generate derived string s(r)
r ← all blocks from S that a
not in S(

s'(r) ← s(r) + m
(Concatenation of s(r) and

fitness(r) ← $\frac{1}{||s'(r)||^2}$

return fitness.

Cross over - individuals (S, P1, P2) {

new [];

Paper Theory

__/__/__

```
for i in range(P1.length):
    p = random.random()
    if p <= 0.5
    New[i] = P1[i];
    elif p <= 0.9
    New[i] = P2[i];
    else
        New[i] = mutate(s)
}
return New;
```

Steps:

1) Initialize population of possible solution.
2) Evaluate fitness - It tells how good the solution is.
3) While termination not reached:

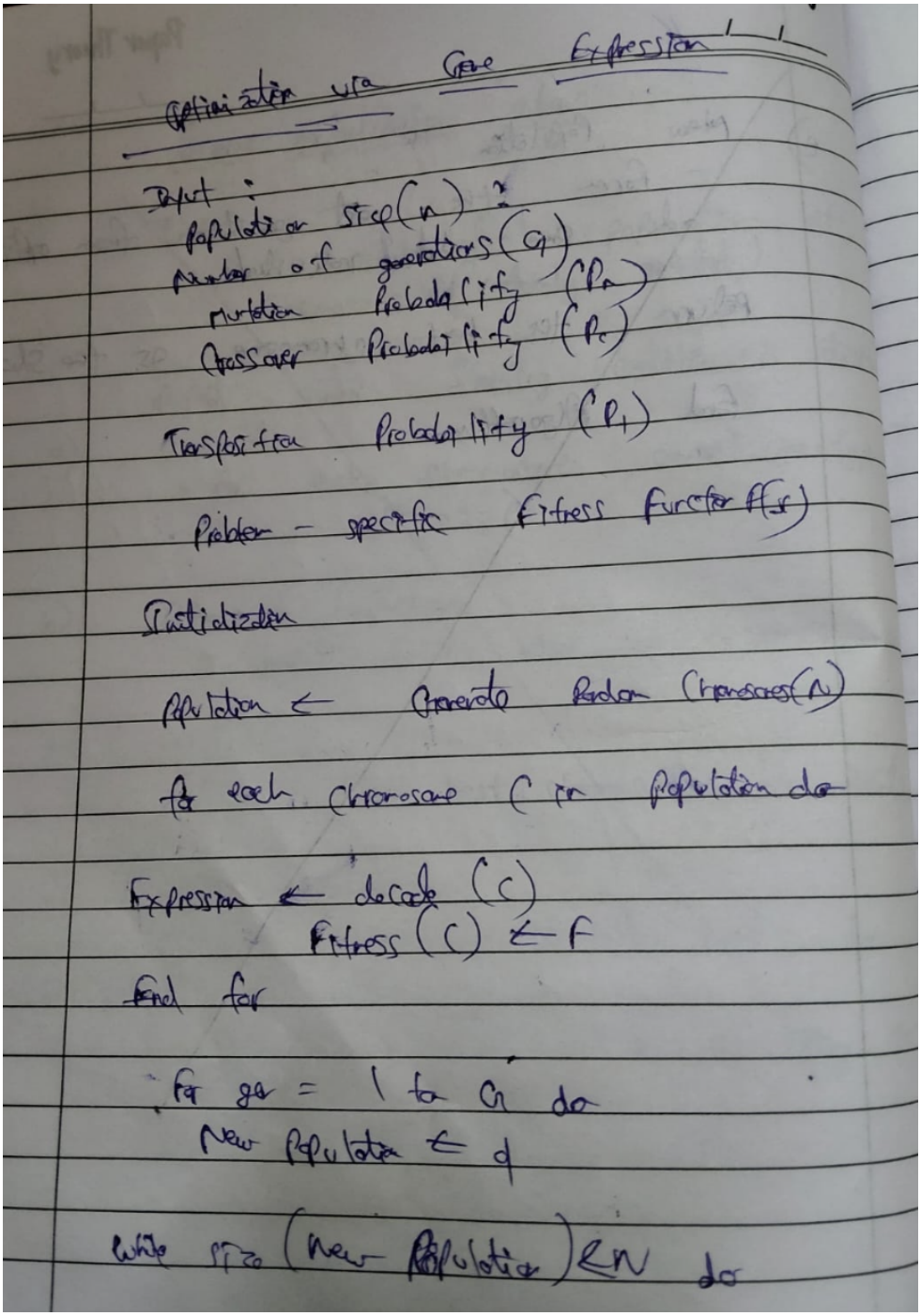
- select individuals from P(t)
- recombine selected individuals
- mutate some individuals
- Evaluate fitness of new individuals
- Create new population

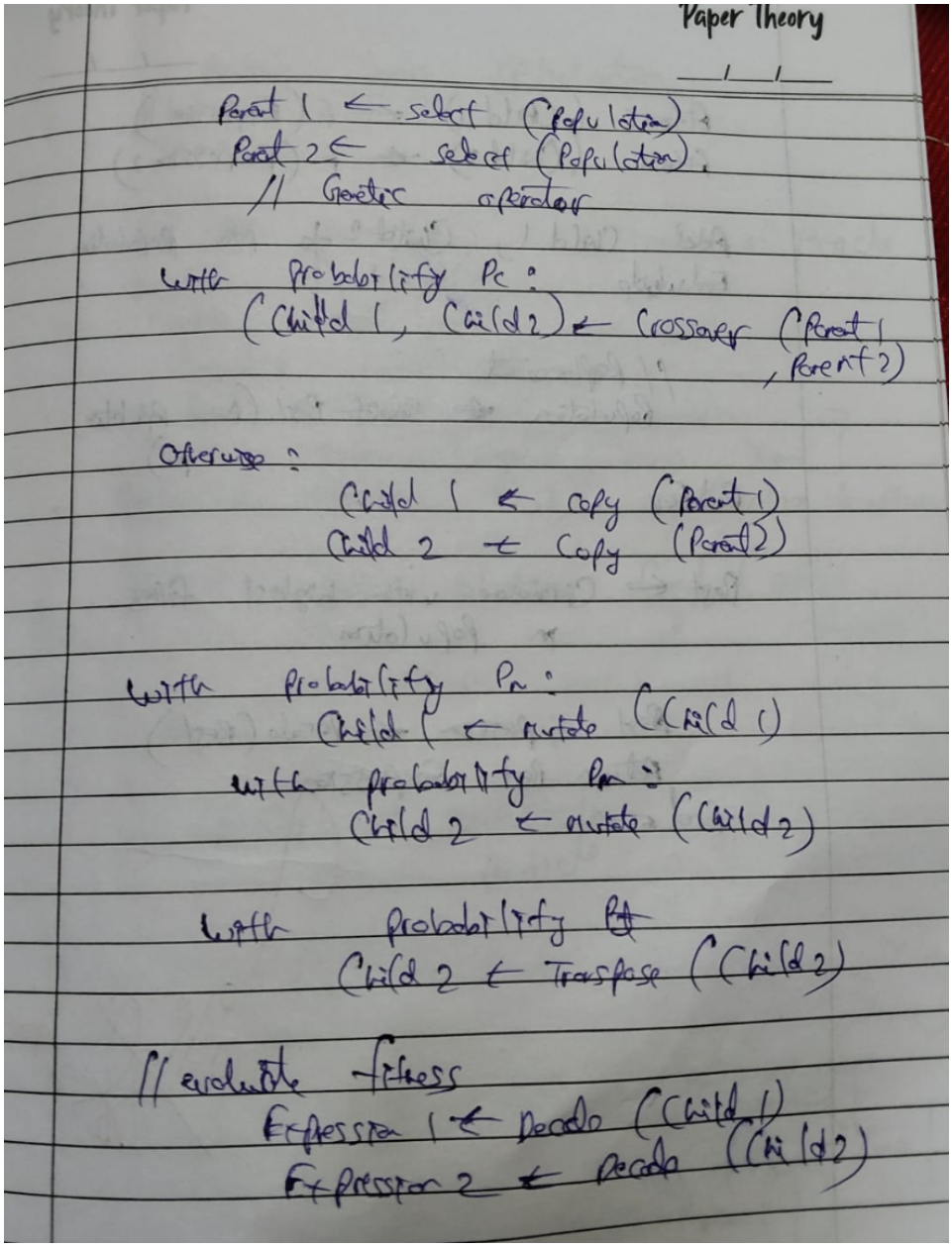
Code :

```
import numpy as np
import random
import matplotlib.pyplot as plt

# target star constellation (fixed points)
target = np.array([
   [1,1], [2,3], [3,2], [4,4], [5,1]
])

pop_size = 30
num_stars = len(target)
generations = 100
mutation_rate = 0.1

def create_individual():
   return np.random.rand(num_stars, 2) * 6

def fitness(ind):
   return np.sum((ind - target)**2)

def selection(pop):
   scores = [(fitness(ind), ind) for ind in pop]
   scores.sort(key=lambda x: x[0])
   return scores[0][1], scores[1][1]

def crossover(p1, p2):
   cut = random.randint(1, num_stars-1)
   c1 = np.vstack((p1[:cut], p2[cut:]))
   c2 = np.vstack((p2[:cut], p1[cut:]))
   return c1, c2

def mutate(ind):
   if random.random() < mutation_rate:
      i = random.randint(0, num_stars-1)
      ind[i] = np.random.rand(2) * 6
   return ind

pop = [create_individual() for _ in range(pop_size)]
```

```
for g in range(generations):
    new_pop = []
    for _ in range(pop_size // 2):
        p1, p2 = selection(pop)
        c1, c2 = crossover(p1, p2)
        new_pop.append(mutate(c1))
        new_pop.append(mutate(c2))
    pop = new_pop

best = min(pop, key=fitness)

plt.scatter(target[:,0], target[:,1], s=80)
plt.scatter(best[:,0], best[:,1], s=80)
plt.title("Genetic Algorithm – Star Constellation Matching")
plt.legend(["Target", "Evolved"])
plt.show()
```

Program 2:
Gene Expression



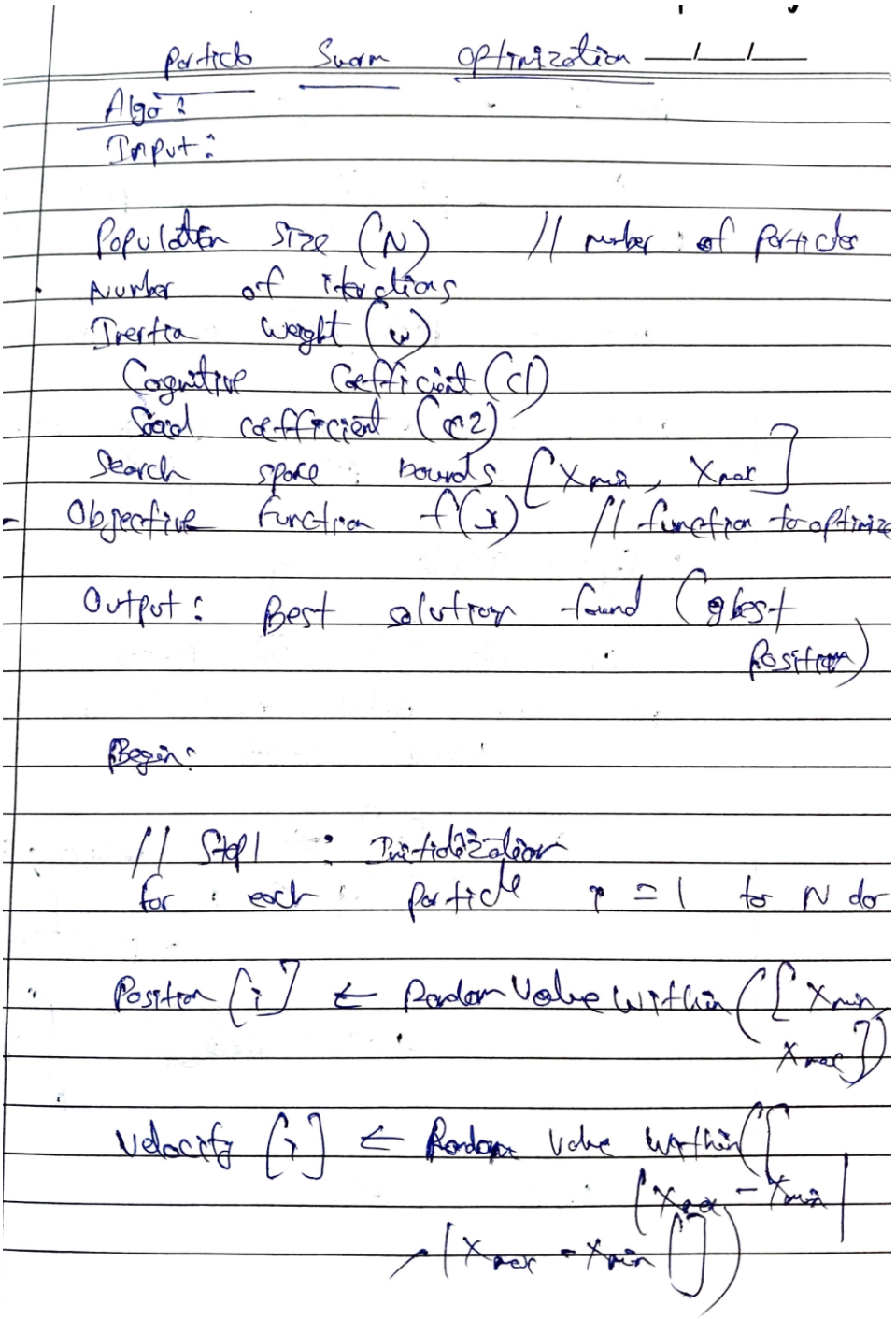

Paper Theory
Parent 1 ← select (Population)
Parent 2 ← select (Population)
// Genetic operator
With probability Pc:
(Child 1, Child 2) ← Crossover (Parent 1, Parent 2)
Otherwise:
Child 1 ← copy (Parent 1)
Child 2 ← copy (Parent 2)
With probability Pm:
Child 1 ← mutate (Child 1)
With probability Pm:
Child 2 ← mutate (Child 2)
With probability Pt
Child 2 ← Transpose (Child 2)
// evaluate fitness
Expression 1 ← Decode (Child 1)
Expression 2 ← Decode (Child 2)

Code:

```
import numpy as np
import random


img = np.random.rand(100,100)  # simple random image


def add(x, y): return x + y
def mul(x, y): return x * y
def sub(x, y): return x - y

ops = [add, mul, sub]
consts = [0.2, 0.5, 1.0]


def random_expr():
    return (random.choice(ops), random.choice(consts), random.choice(consts))


def apply_expr(expr, img):
    op, a, b = expr
    return np.clip(op(img * a, img * b), 0, 1)


def fitness(expr):
    new_img = apply_expr(expr, img)
    return new_img.std()


population = [random_expr() for _ in range(20)]

for gen in range(10):
    scored = sorted(population, key=fitness, reverse=True)
    population = scored[:10]  # select top 10

    # mutate
    for i in range(10):
        if random.random() < 0.3:
```

```
            population.append(random_expr())

    print("Generation", gen, "Best fitness:", fitness(population[0]))

best = population[0]
print("\nBest evolved expression:", best)
```

Program 3:
Particle Swan Optimization



Particle Swarm optimization ___/___/___

Algo:

Input:

Population size (N) // number of particles
Number of iterations
Inertia weight (w)
Cognitive coefficient (c1)
Social coefficient (c2)
Search space bounds [x_{min}, x_{max}]
Objective function f(x) // function to optimize

Output: Best solution found (gbest position)

Begin:

// Step1 : Initialization
for each particle i = 1 to N do

Position [i] ← Random value within ([x_{min}, x_{max}])

Velocity [i] ← Random value within ([$-|x_{max} - x_{min}|$, $|x_{max} - x_{min}|$])

pbest[i] ← position[i]

fitness_pbest(i) ← f(position[i])

End for

gbest ← Best position among pbest

fitness_gbest ← Best ~~position~~ fitness among fitness_pbest

// Step2 : Iteration Loop

For iter = 1 to Maxiter do

for each particle i=1 to N do

update velocity

velocity[i] ← w * velocity[i]

+c1 * rand() * (pbest[i] - position[i])

+c2 * rand() * (gbest - position[i])

update position:

gbest ← Position (i)
fitness_gbest ← fitness
End if
End for

Print Progress :

Print ("Iteration", iter, "Best fitness", fitness_gbest)

End for

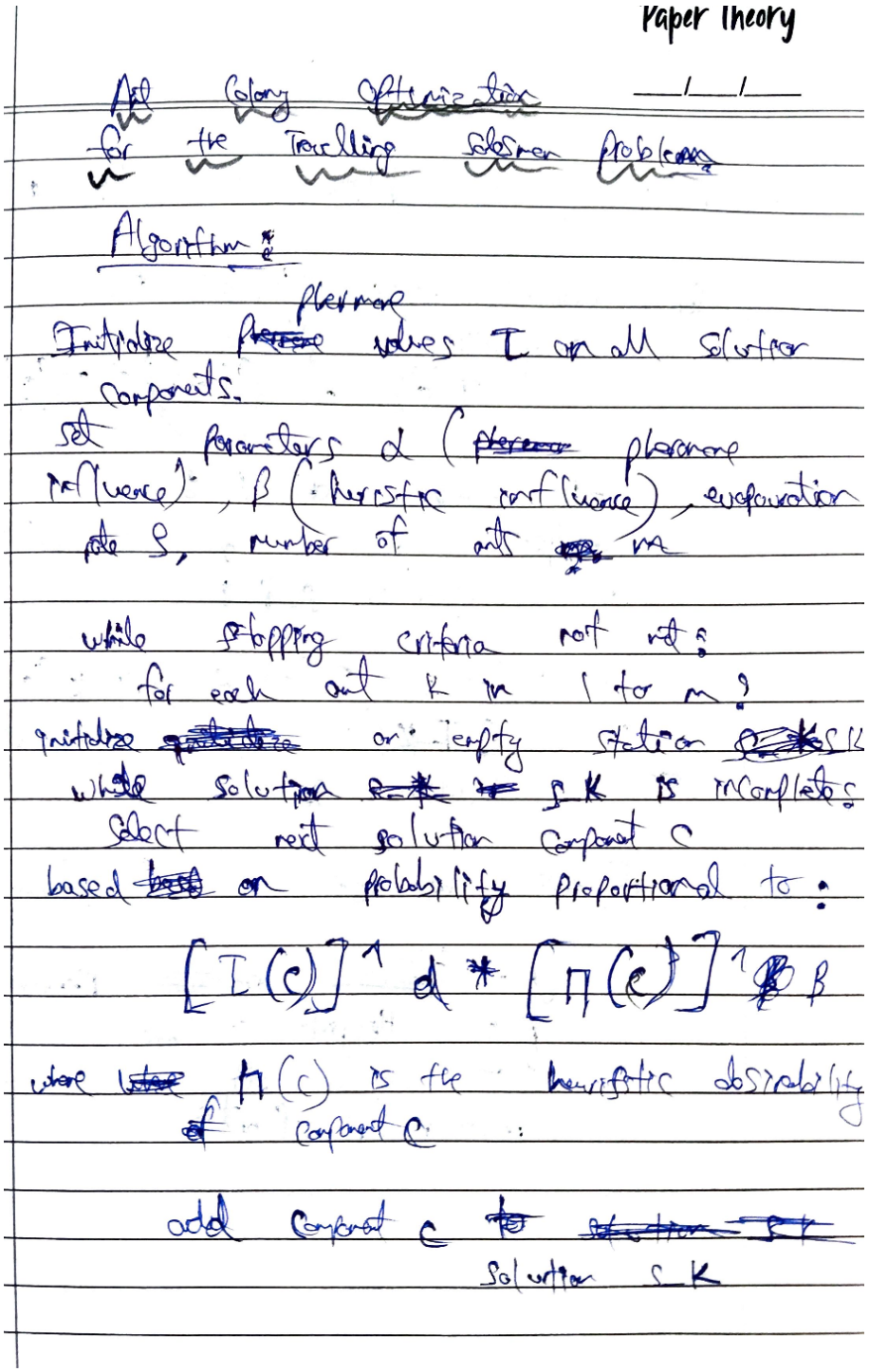
Return best solution
return gbest
End

- Execute in "Energy optimization"
- Explain gbest, pbest, "fitness fun"
- How is "fitness fun" selected & used.

Code:

```
import numpy as np

def energy(x):
    return 0.5*x**2 + 10*np.sin(x) + 20

n=20
w=0.6
c1=1.4
c2=1.4
it=40

x=np.random.uniform(-10,10,n)
v=np.zeros(n)
p=x.copy()
pv=np.array([energy(i) for i in x])
g=p[np.argmin(pv)]
gv=min(pv)

for _ in range(it):
    r1=np.random.rand(n)
    r2=np.random.rand(n)
    v=w*v + c1*r1*(p-x) + c2*r2*(g-x)
    x=x+v
    vals=np.array([energy(i) for i in x])
    b=vals<pv
    p[b]=x[b]
    pv[b]=vals[b]
    if min(vals)<gv:
        g=x[np.argmin(vals)]
        gv=min(vals)

print("Best value:", g)
print("Min energy:", gv)
```

Program 4:
Ant Colony Optimization



Paper Theory

__/__/____

Ant Colony Optimization for the Travelling Salesman problem

Algorithm :

Initialize pheromone values T on all solution components.

Set parameters α (pheromone influence), β (heuristic influence), evaporation rate ρ, number of ants m

while stopping criteria not met:
  for each ant k in 1 to m:
    initialize or empty solution S_k
    while solution S_k is incomplete:
      Select next solution component c based on probability proportional to:

$$[I(c)]^{\alpha} * [\eta(c)]^{\beta}$$

where $\eta(c)$ is the heuristic desirability of component c.

add component c to solution S_k

Evaluate the quality of solution

for each solution Construct e:
evaporate pheromone
$T(e) = (1-\rho) * T(e)$

for each ant k:
deposit pheromone on components
in solution S_k:

$T(e) = T(e) + \Delta T_k(e)$
amount $\Delta T_k(e)$ depends on quality
of solution S_k

returns the best solution found

α - ~~Pheromone influence~~ Pheromone influence
β - Heuristic influence
ρ - Evaporation rate
m - number of ants

Paper theory

__/__/__

$\Delta T_k(c)$ → Reward signal (that guides future ants towards better solution)

[ΔT → Change in Pheromone]

water flow Management

+

TSP

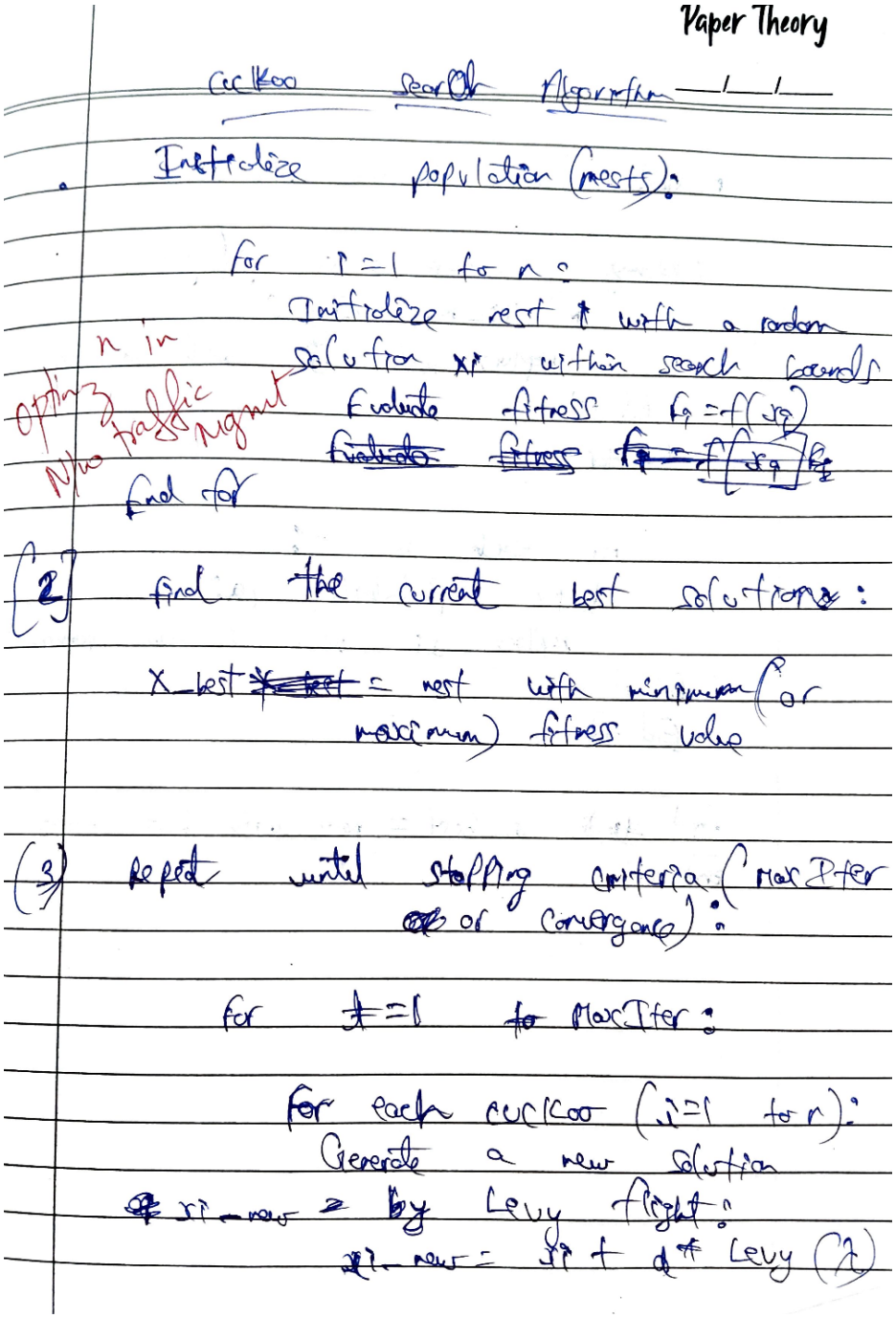
$30/10

Code:

```
import numpy as np
import random

dist=np.array([
    [0,12,10,19,8],
    [12,0,3,7,2],
    [10,3,0,6,20],
    [19,7,6,0,4],
    [8,2,20,4,0]
])

n=dist.shape[0]
ants=20
alpha=1
beta=2
rho=0.5
Q=100
iters=40
pher=np.ones((n,n))

def tour_len(t):
    s=0
    for i in range(n-1):
        s+=dist[t[i],t[i+1]]
    s+=dist[t[-1],t[0]]
    return s

best=None
best_len=1e9

for it in range(1,iters+1):
    all_tours=[]
    for k in range(ants):
        t=[random.randint(0,n-1)]
        for _ in range(n-1):
            c=t[-1]
            probs=[]
            for j in range(n):
```

```
            if j not in t:
                p=(pher[c,j]**alpha)*((1/dist[c,j])**beta)
                probs.append((j,p))
        s=sum([p for _,p in probs])
        r=random.random()*s
        tot=0
        for j,p in probs:
            tot+=p
            if tot>=r:
                t.append(j)
                break
    all_tours.append(t)

for t in all_tours:
    L=tour_len(t)
    if L<best_len:
        best=t
        best_len=L

pher=(1-rho)*pher
for t in all_tours:
    L=tour_len(t)
    for i in range(n-1):
        a,b=t[i],t[i+1]
        pher[a,b]+=Q/L
        pher[b,a]+=Q/L
    a,b=t[-1],t[0]
    pher[a,b]+=Q/L
    pher[b,a]+=Q/L

print("Iteration",it,"| Best:",best_len)

print("Final Best Path:",best)
print("Final Best Length:",best_len)
```

Program 5:
Cuckoo Search Algorithm

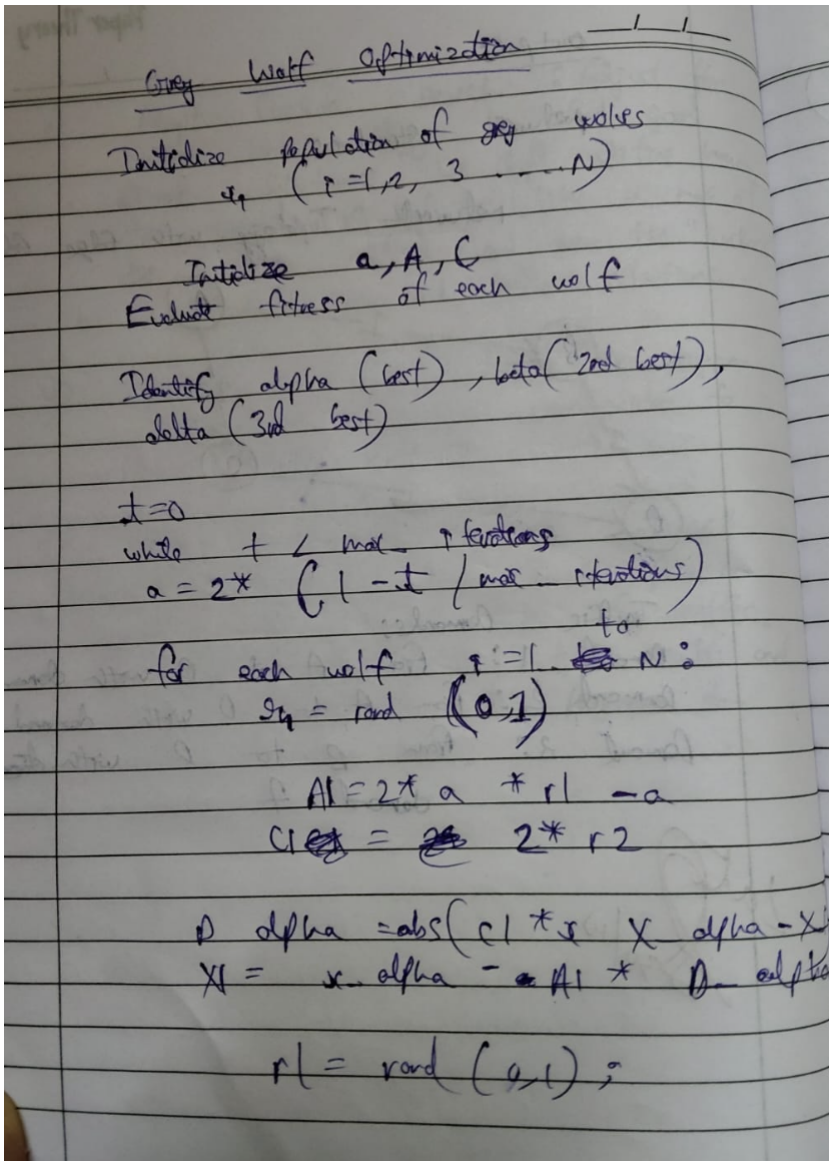


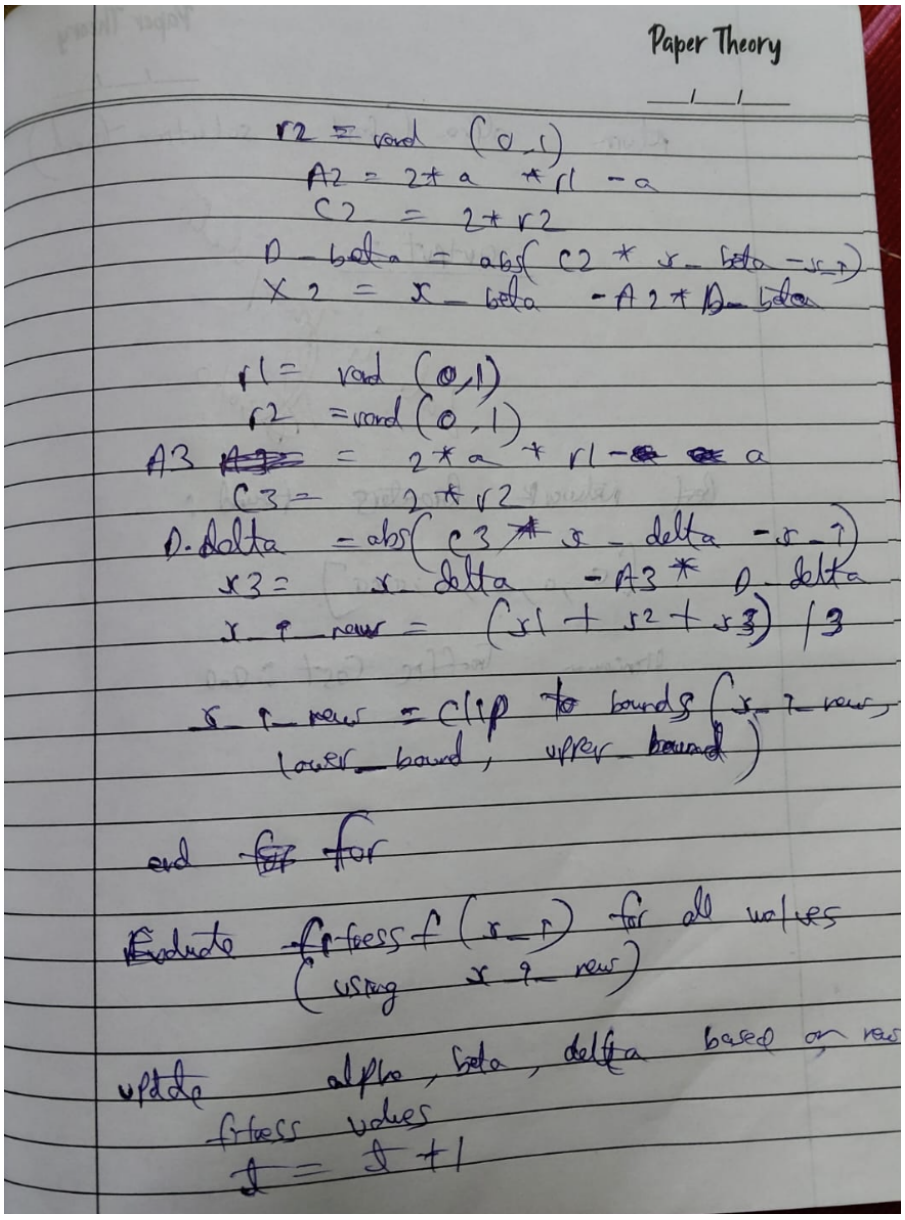
Code:

```
import numpy as np
import random

def cost(x):
    return 0.5*x**2 + 20*np.sin(x) + 30

n=20
lb=-10
ub=10
pa=0.25
iters=40

nest=np.random.uniform(lb,ub,n)
fitness=np.array([cost(x) for x in nest])

def levy():
    u=np.random.normal(0,1)
    v=np.random.normal(0,1)
    s=u/abs(v)**0.5
    return s

best=nest[np.argmin(fitness)]
best_val=min(fitness)

for it in range(1,iters+1):
    for i in range(n):
        step=levy()
        new=nest[i]+step*(nest[i]-best)
        new=np.clip(new,lb,ub)
        if cost(new)<fitness[i]:
            nest[i]=new
            fitness[i]=cost(new)

    for i in range(n):
        if random.random()<pa:
            j=random.randint(0,n-1)
            k=random.randint(0,n-1)
            new=nest[i]+random.random()*(nest[j]-nest[k])
```

```
            new=np.clip(new,lb,ub)
            if cost(new)<fitness[i]:
                nest[i]=new
                fitness[i]=cost(new)

    best=nest[np.argmin(fitness)]
    best_val=min(fitness)
    print("Iteration",it,"| Best:",best_val)

print("Final Best Traffic Load:",best)
print("Final Congestion Cost:",best_val)
```

Program 6:
Grey wolf Optimization




Paper Theory
r2 = rand (0,1)
A2 = 2*a *r1 - a
C2 = 2*r2
D_beta = abs(C2 * x_beta - x_i)
X2 = x_beta - A2 * D_beta
r1 = rand (0,1)
r2 = rand (0,1)
A3 = 2*a * r1 - a
C3 = 2*r2
D_delta = abs(C3 * x_delta - x_i)
x3 = x_delta - A3 * D_delta
x_i_new = (x1 + x2 + x3) / 3
x_i_new = clip to bounds (x_i_new, lower_bound, upper_bound)
end for
Evaluate fitness f(x_i) for all values (using x_i_new)
update alpha, beta, delta based on new fitness values
t = t + 1

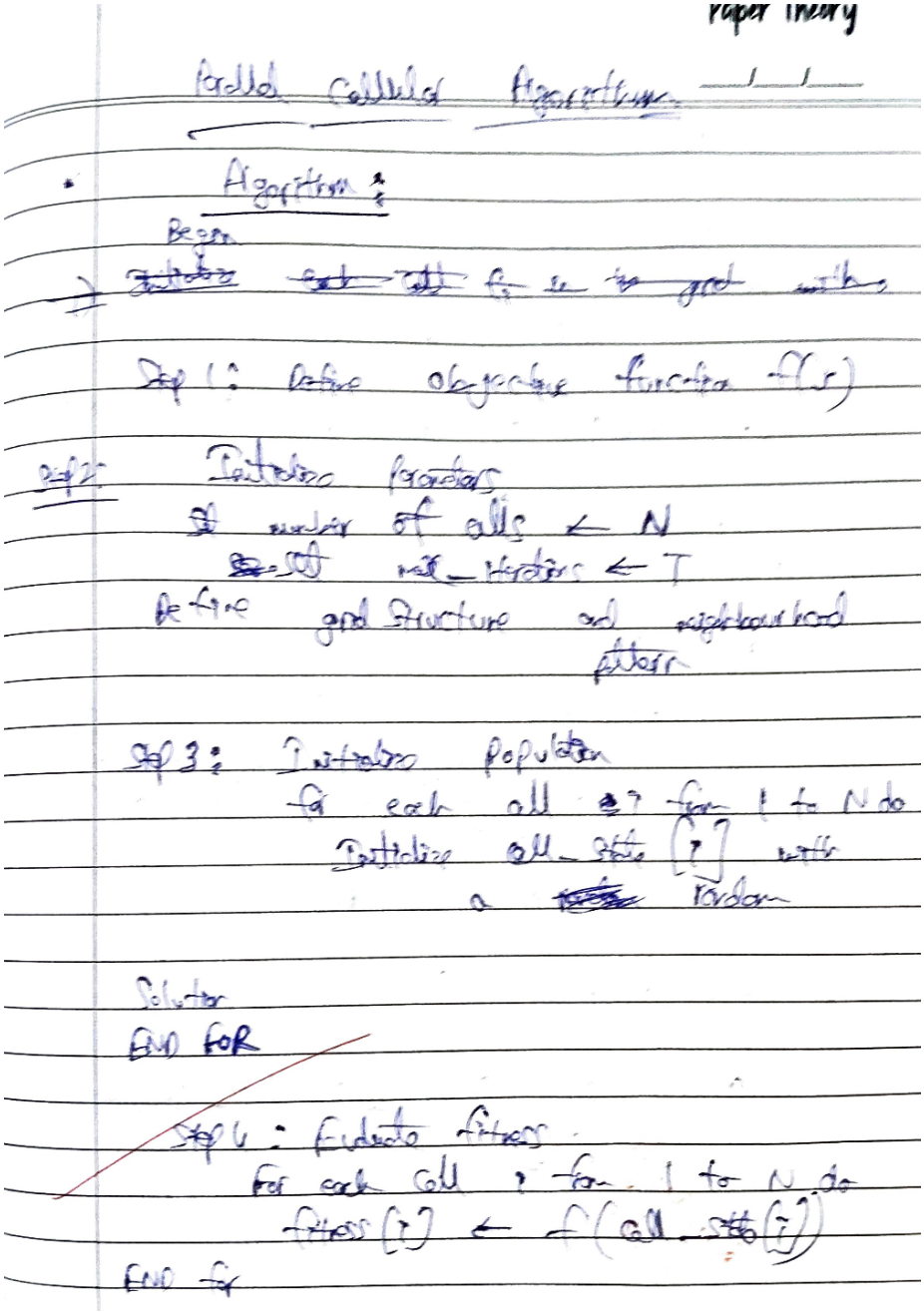
Code:

```
import numpy as np
import time

# Objective function (Sphere)
def f(x):
    return np.sum(x**2)

# Grey Wolf Optimizer
def GWO(dim=10, search_agents=20, max_iter=100):
    lb, ub = -10, 10
    positions = np.random.uniform(lb, ub, (search_agents, dim))

    # α, β, δ wolves
    Alpha_pos = np.zeros(dim)
    Beta_pos = np.zeros(dim)
    Delta_pos = np.zeros(dim)

    Alpha_score = float("inf")
    Beta_score = float("inf")
    Delta_score = float("inf")

    early_stop = int(max_iter * 0.6)  # incomplete execution (stop at 60%)

    for t in range(max_iter):
        for i in range(search_agents):
            positions[i] = np.clip(positions[i], lb, ub)
            fitness = f(positions[i])

            if fitness < Alpha_score:
                Alpha_score = fitness
                Alpha_pos = positions[i].copy()

            elif fitness < Beta_score:
                Beta_score = fitness
                Beta_pos = positions[i].copy()

            elif fitness < Delta_score:
                Delta_score = fitness
```

```
            Delta_pos = positions[i].copy()

        a = 2 - t * (2 / max_iter)

        for i in range(search_agents):
            for j in range(dim):
                r1, r2 = np.random.rand(), np.random.rand()
                A1 = 2 * a * r1 - a
                C1 = 2 * r2
                D_alpha = abs(C1 * Alpha_pos[j] - positions[i][j])
                X1 = Alpha_pos[j] - A1 * D_alpha

                r1, r2 = np.random.rand(), np.random.rand()
                A2 = 2 * a * r1 - a
                C2 = 2 * r2
                D_beta = abs(C2 * Beta_pos[j] - positions[i][j])
                X2 = Beta_pos[j] - A2 * D_beta

                r1, r2 = np.random.rand(), np.random.rand()
                A3 = 2 * a * r1 - a
                C3 = 2 * r2
                D_delta = abs(C3 * Delta_pos[j] - positions[i][j])
                X3 = Delta_pos[j] - A3 * D_delta

                positions[i][j] = (X1 + X2 + X3) / 3

        # Late execution simulation
        time.sleep(0.15)

        print(f"Iter {t+1} | Alpha Score: {Alpha_score:.6f}")

        # Stop early (incomplete execution)
        if t == early_stop:
            print("\n       Early stopping triggered (incomplete execution)\n")
            break

    return Alpha_pos, Alpha_score


# Run GWO
```

```
best_pos, best_score = GWO(dim=10, search_agents=20, max_iter=100)
print("\nBest Found Position:", best_pos)
print("Best Score:", best_score)
```

Program 7:
Parallel Cellular Algorithm



Iteration ← 0

for each cell i from 1 to N do
PARALLEL
Identify Neighbour Set (i)
new_state[i] ← update Rule [cell_state[i]
Neighbour set (i)]
END for

State (Synchronous update)
for each cell i from 1 to N do
cell_state[i] ← new_state[i]
fitness[i] ← f (cell_state[i])
END for
Iteration ← Iteration + 1

END WHILE

Output cell_state[K] as Best_solution
END

Routing & Scheduling

Code:

```
import numpy as np

n=12
m=3
tasks=np.random.randint(5,25,n)
routes=[0]*m
assign=[[] for _ in range(m)]

for t in tasks:
    r=np.argmin(routes)
    routes[r]+=t
    assign[r].append(t)

print("Tasks:",tasks)
print("Route Loads:",routes)
print("Assignments:",assign)
print("Final Makespan:",max(routes))
```