# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.



### LAB RECORD

## **Bio Inspired Systems (23CS5BSBIS)**

Submitted by

Sindhuja Narasimhan (1BM22CS279)

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
Sep-2024 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 **Sindhuja Narasimhan (1BM22CS279)**, 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.

Saritha A.N Assistant Professor Department of CSE, BMSCE Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

## Index

Sl. No.	Date	<b>Experiment Title</b>	Page No.
1	9-10-24	Genetic Algorithm	1-4
2	16-10-24	Ant Colony Optimization	5-7
3	30-10-24	Particle Swarm Optimization	8-12
4	13-11-24	Cuckoo Search Algorithm	13-15
5	20-11-24	Grey Wolf Optimizer	16-19
6	27-11-24	Parallel Cellular Algorithm	20-23
7	4-12-24	Gene Expression Algorithm	24-27

Github Link:

https://github.com/sindhuja279/BISlab.git

Program 1
Genetic Algorithm for Optimization Problems

1	classmate
03/10/24	classmate  Date Page 3
1	Lab 2.
1	Algorithm for genetic algorithm
Step 1.	Inhalize parameter: set du population eize, medation rale, cuestoure evate & no of generations.
Зкрг.	Generale Initial population: create a handon population of potential selections within the given bourds.
	Evaluate Finess: calculate the finess of each undividual in the population by evaluating the objective function.
	Select Parent: Eelect the fittess individuals from the population do supresque kased on this
	Crossoves perform crossover between the selected parents to creat new effspring, with a producting equal to the crossover tall
Step 6.	Midate: apply mitation to the affspring with a probability agral to mutator state, to introduce new trails
Step 7.	Repley Least Fit: ruplas de least fit individuals in the population with the new Appring
	generation on with consequent curio and met.
Stop 9.	Supply Best Selo: Meturo the best solo found during the generation, which is the individual water the

7 The basic idea behind a genetic simulate the process if evolution selecting, breeding and mutaling candidate solution. > The genetic algorithm is used for · Oprimizing puchent such as scheduling, resource all and pointatio apprintation machine learning to optimize parameters of such as newal network and olecision trees · enginering designs: to optimize complex systems electronic circuit etc: · computer network: to optimize network scourting and resource allocation. -> Optimizing Techniques: · selection: selecting the fittest andividuals from populator to suproduce orosseres: combining the generic information posent to create a new approprio mutation: Mandomly charging the genetic information an individual do introduce new traiteslitism: preserving the best solution officer the generation do ensure that the best solution are not · Simulated bringly crossover & crossover technique that simulas the prouse of binary crossors to create

```
Code:
import random
import numpy as np
def objective function(x):
  # Define the objective function to be minimized
  return x^{**}2 + 2^*x + 1
def generate_initial_population(population_size, bounds):
  # Generate an initial population of potential solutions
  population = []
  for _ in range(population_size):
     x = random.uniform(bounds[0], bounds[1])
     population.append(x)
  return population
def evaluate_fitness(population):
  # Evaluate the fitness of each individual in the population
  fitness = []
  for x in population:
     fitness.append(objective_function(x))
  return fitness
def selection(population, fitness, num_parents):
  # Select individuals based on their fitness to reproduce
  parents = []
  for _ in range(num_parents):
     max_fitness_idx = np.argmax(fitness)
     parents.append(population[max_fitness_idx])
     fitness[max fitness idx] = -float('inf') # Set fitness to -inf to avoid re-selection
  return parents
def crossover(parents, crossover_rate):
  # Perform crossover between selected individuals to produce offspring
  offspring = []
  for _ in range(len(parents) // 2):
     parent1, parent2 = random.sample(parents, 2)
     if random.random() < crossover rate:
       x1, x2 = parent1, parent2
       x offspring = (x1 + x2)/2
       offspring.append(x_offspring)
  return offspring
def mutation(offspring, mutation_rate, bounds):
```

# Apply mutation to the offspring to maintain genetic diversity

for i in range(len(offspring)):

```
if random.random() < mutation rate:
       offspring[i] += random.uniform(-0.1, 0.1) * (bounds[1] - bounds[0])
       offspring[i] = max(bounds[0], min(offspring[i], bounds[1])) # Ensure bounds
  return offspring
def genetic_algorithm(population_size, mutation_rate, crossover_rate, num_generations, bounds):
  population = generate_initial_population(population_size, bounds)
  for generation in range(num_generations):
    fitness = evaluate fitness(population)
    parents = selection(population, fitness, population_size // 2)
    offspring = crossover(parents, crossover rate)
    offspring = mutation(offspring, mutation_rate, bounds)
    population = offspring + parents
  best_solution = min(population, key=objective_function)
  return best solution
# Set parameters
population_size = 100
mutation rate = 0.1
crossover\_rate = 0.5
num generations = 100
bounds = (-10, 10)
# Run the Genetic Algorithm
best_solution = genetic_algorithm(population_size, mutation_rate, crossover_rate, num_generations,
bounds)
print("Best solution:", best_solution)
```

Program 2
Particle Swarm Optimization for Function Optimization

Classnate Date Page	classmate Dole Prigo 9
pollisher  2 pso is an oppinization algorithm inspected by the  sound behaviour of animals such as that flacery or  fish schoolige The goal is to find appined ear many  position adjust to complex problems by having muly  particle explore the adjuster space. Each particle  acquired sits politic trased on sits cover experient	sef cognina coeff , cognina coeff sef sacel coeff , sacel coeff sef max let , movies sef gwan, Claires (dem) for in rage (nun paintelle) sef gwan, best posson, sef awam [0], trest posson seff global his value & seff swam [0], trest value def approved (self): for iteration in range (sef max ites):
conveying towards the Dest solution	for particle in self swam:
is Algorithma Cool Implementation	painer ber value = from value
import numpy as np	pouriel trust person, rp. capy/ princle position
class Buticle:  def int (sex, aim);  self position of random uniform (-5.12, dim)  get velouty of up. sandom uniform (-1, dim)	gett. global best value. best value.  gett. global best value. best value  auf global best value. best value  position;  for particle in self-swarm;  gur, np. vandom; self. dim)  yz, np. vandom; self. dim)  particle. velously, (self. inertic weight + particle. velously, coeff to a (particle best position)
self. Best position. op. copy (self. position) self. Best value. Flasteligen function (self. position) class PSD:	sef. soval weff + 22 (sef. global-but pas))
self. num particles: num particles  self. num particles: num particles  self. aim; dim  self. inoution weight: inche ways	particle point 12 parties, self. global best value

### Code: import numpy as np # Define the Rastrigin function def rastrigin\_function(x): A = 10return A \* len(x) + sum( $x_i**2 - A * np.cos(2 * np.pi * x_i)$ for x\_i in x) # Particle Swarm Optimization Algorithm class Particle: def \_\_init\_\_(self, dim): self.position = np.random.uniform(-5.12, 5.12, dim) # Random position self.velocity = np.random.uniform(-1, 1, dim) # Random velocity self.best\_position = np.copy(self.position) # Personal best position self.best\_value = rastrigin\_function(self.position) # Personal best value class PSO: def \_\_init\_\_(self, num\_particles, dim, inertia\_weight, cognitive\_coeff, social\_coeff, max\_iter): self.num\_particles = num\_particles self.dim = dimself.inertia\_weight = inertia\_weight self.cognitive coeff = cognitive coeff self.social\_coeff = social\_coeff self.max iter = max iter self.swarm = [Particle(dim) for \_ in range(num\_particles)] self.global best position = self.swarm[0].best position self.global\_best\_value = self.swarm[0].best\_value def optimize(self): for iteration in range(self.max\_iter): for particle in self.swarm: # Evaluate fitness fitness\_value = rastrigin\_function(particle.position) # Update personal best if fitness value < particle.best value: particle.best\_value = fitness\_value particle.best\_position = np.copy(particle.position) # Update global best if fitness value < self.global best value: self.global\_best\_value = fitness\_value self.global\_best\_position = np.copy(particle.best\_position) # Update velocities and positions for particle in self.swarm: r1 = np.random.rand(self.dim)

```
r2 = np.random.rand(self.dim)
          # Update velocity
          particle.velocity = (self.inertia_weight * particle.velocity +
                       self.cognitive_coeff * r1 * (particle.best_position - particle.position) +
                       self.social_coeff * r2 * (self.global_best_position - particle.position))
          # Update position
          particle.position += particle.velocity
     return self.global_best_position, self.global_best_value
# Parameters
num_particles = 30 # Number of particles
dim = 2 # Dimensionality of the problem
inertia_weight = 0.7 # Inertia weight
cognitive_coeff = 1.5 # Cognitive coefficient
social_coeff = 1.5 # Social coefficient
max_iter = 100 # Maximum number of iterations
# Run PSO
pso = PSO(num_particles, dim, inertia_weight, cognitive_coeff, social_coeff, max_iter)
best_position, best_value = pso.optimize()
print("Best Position:", best_position)
print("Best Value:", best_value)
```

**Program 3**Ant Colony Optimization for the Traveling Salesman Problem

	classmate
	Classmate Date Page M
4/11/04	Page 11
mp.	
	aab:5 Ant Colony approvation
	dab:5 Ant colony apringation
	And colony continuous sounds as times in
	And colony cophingation prevides a biologically ciripinal
1000	Il sowie the beautiline falows with a
1 1 to	There is the material behavior of not a
104.00	The state of the paper of the
	and the explore the solution
	space and conveye downed exprimal of near-grim
	eductions for TSP.
	(asymptotic ) and one ways of a
	Impat Church at an
- C	impart numpy as np
	import handon
	THE MANUFACTURE OF THE PROPERTY OF THE PROPERT
	class AntColony:
	def init Coly, cities, num and appear, next, sine,
	iterations):
	good cities cities
Carries"	self cities, cities
	enf num cities len (cities)
	self num and num and
	ay ayra, ayra
	sey, beto, beto
	Cilf. elho, gino
	sef bust evoids. Mone
	self. best distances float ('inf')
	Not calculate distante and the second
er that	def calculate distant matrix (seef):
	distance moder on zeros (( self num cities, self in cons)
	for 1 in songe (self num cities);
	for Jun sarge (self. num cities);
	distance, mater Is ILiJ . mp. linalg. now (rp. and
	elf cities tis- mp anay ( eng cities (; 1))
	seetuin distani mateur.
14mm	ail succelled beginness the Tilled ber
	May de la

· ola-	
Date	classmate
Page 49	Date Poge 13
40	11-11-12
def charge vext confeely, arent city, visited):  probabilities of for city in eagle (self num cities):	def coptimie (ery):
popapilitael	all in second sell its instead;
las vitu ain ename (self mum citics):	Got in starget self iterations!
y city not in visked	gety construct solutor()
premone self phenon Coursed any Ilay 7 - see	ween seek past seowy seek best distance
howing ( I / Jey, dis some most awers city] tary - 20	The state of the s
pheumone, left pheumone Coursest ity I (ity ] " seff of heuristic (I sey dis sone most course city I city I expend probabilities append (pheumone * heuristic)	if name : "main":
granium spy cy	Cites (C(0,0), (1,2), (2,4)(3,1), (a, 2)
groba intrus apperalo	. mum anti co
(MODALUTING) (MACAMINA)	
gendebilines. mp. anayl plobabilion	alpro 11:0
plobabilities / 1 probability: Eurol)	pero : 1.0 pero : 2.0 pero : 0.5
Reten, np. sandan. choi ectarge deg. then cho p?	9.00 1 0.3
def construct solution (self):	iksotrom 10
def construct solution (self):	aw. Antidopy (lites, num and, alpho, tola, the, ileston)
for in eager lef num and:	bust route, ass distance. are official)
for in enargetelf-num and:	print ( Best poids; best words)
for in Harpel, self, mun cities):	print ( Best Distant; best Obiland).
next city : Self. Choose next city ( Usted [ ], Visited)	Carrello CC (Olabor al
visika append (snext uty)	I we still a major a parador allowed a first of a
visited appendico)	Ocuput:
stitute on soll soll soll water relations (Visited)	( sery / 2 1/2 des they record de tre due
distant : sey calmente noute austana (Visited)	Best Roid; [0.1, 2, a, 3, 0]
if distance self best distance:	Part Nick 10, 10 (50940
Self. best- clistane, distance	Out autono. The same of the sa
sey best nout : writed,	The section of the se
	Company and the aid on the manual specific mentals
def calculate route distance (self, seoute):	The second second
retur sum ( set, distant matrix ( noverill reacte list	CONTRACTOR OF THE PROPERTY OF
for 1 us saye (len ( soute )-17)	All fraction transport of:
the same to the same shall	(and a larisoning grains
de upage phemone (sex):	
seeff. premore v. Ci-sey, eno)	and the second control with
101 coly 10 (2000 CO) 1000	and the second wife it, private,
Got cay in pronge clen (self test next) - ):	the second contract forms
3cf. phenone (lef. best rout (City IIIsef. kess south	was for the same of the same o
1 sef best-dispone.	64,298

### Code:

```
import numpy as np
import random
class AntColony:
  def __init__(self, cities, num_ants, alpha, beta, rho, iterations):
     self.cities = cities
     self.num_cities = len(cities)
     self.num_ants = num_ants
     self.alpha = alpha
     self.beta = beta
     self.rho = rho
     self.iterations = iterations
     self.pheromone = np.ones((self.num_cities, self.num_cities)) # Initial pheromone levels
     self.distance_matrix = self.calculate_distance_matrix()
     self.best route = None
     self.best_distance = float('inf')
  def calculate_distance_matrix(self):
     distance_matrix = np.zeros((self.num_cities, self.num_cities))
     for i in range(self.num_cities):
       for j in range(self.num_cities):
          distance_matrix[i][j] = np.linalg.norm(np.array(self.cities[i]) - np.array(self.cities[j]))
     return distance_matrix
  def choose_next_city(self, current_city, visited):
     probabilities = []
     for city in range(self.num_cities):
       if city not in visited:
          pheromone = self.pheromone[current_city][city] ** self.alpha
          heuristic = (1 / self.distance_matrix[current_city][city]) ** self.beta
          probabilities.append(pheromone * heuristic)
       else:
          probabilities.append(0)
     probabilities = np.array(probabilities)
     probabilities /= probabilities.sum() # Normalize
     return np.random.choice(range(self.num_cities), p=probabilities)
```

```
def construct_solution(self):
     for _ in range(self.num_ants):
       visited = [0] # Start from the first city
       for _ in range(1, self.num_cities):
          next_city = self.choose_next_city(visited[-1], visited)
          visited.append(next_city)
       visited.append(0) # Return to the starting city
       distance = self.calculate route distance(visited)
       if distance < self.best_distance:
          self.best distance = distance
          self.best_route = visited
  def calculate_route_distance(self, route):
     return sum(self.distance_matrix[route[i]][route[i + 1]] for i in range(len(route) - 1))
  def update_pheromones(self):
     # Evaporate pheromones
     self.pheromone *= (1 - self.rho)
     # Add pheromones based on the best route found
     for city in range(len(self.best_route) - 1):
       self.pheromone[self.best_route[city]][self.best_route[city + 1]] += 1 / self.best_distance
  def optimize(self):
     for _ in range(self.iterations):
       self.construct_solution()
       self.update_pheromones()
     return self.best_route, self.best_distance
# Example usage:
if __name__ == "__main__":
  # Define cities as (x, y) coordinates
  cities = [(0, 0), (1, 2), (2, 4), (3, 1), (4, 3)]
  # ACO parameters
  num_ants = 10
  alpha = 1.0 # importance of pheromone
  beta = 2.0 # importance of heuristic
```

```
rho = 0.5 # pheromone evaporation rate
iterations = 10

aco = AntColony(cities, num_ants, alpha, beta, rho, iterations)
best_route, best_distance = aco.optimize()

print("Best Route:", best_route)
print("Best Distance:", best_distance)
```

Program 4 Cuckoo Search (CS)

and	CIRCEPTOR LE
21-11-29 Lab 6: Lucuso Search	On the second se
Lab 6: Luction states.  Cuckoo stant is a imprised application algorithm of some cureo production of some cureo species. This personal circolar laying aggs in the trial, leading to the trial, leading to the trial, and	Under 1-2 ave (Course a michage) to mark in
round application along	finess = np. array (( finess-function (ness) for nest in
Cuchoo search us o way program & some gouthon	hest ide , on grown (library)
band in the most princolves laying con	best idx. op. argmin (filmer) best, mesti mesti best idx]
species. This perainer tirds leading of	heat- finess; finess [best-lar]
in next of white strong CS was	and the second second
apprintation of serviced Absolutions, CS cless Lein	for iterator in range (maxiter):
appinization of security new educing proming global security search ocapabilities and avoraing local minime	601; in Evang e (n. rusk):
seach capabilities and accept the manime.	sep- luy festila, or dim
	new ness nesstid+ 3+p+ (nesstid- best nest)
import numpy as not import sandon	new ness one coipl new ness, lower bound,
import evandom	new near np. cap (new near, lower bound)
from sup, special import gommo	new-firmes. fitness function (new nest)
	of new faners & farress (1).
def luy fregor (beda + 5, d : 1):	nestil new-nest
signo ui np. pouce(Cgammo (1+ beta) + pp. sin (pp.)	finessis. new finess.
signe 11 np. power(Egamo (1+ beta) + np. sin (npp.  beta/o)/gama((1+ beta)/o) + beta * np. ca(np	if new finess & best fitness:
pi * peta/e) * + 2), 1/beta)	Dest nest new nest
el. np. vanden. normall o, ligna u, lixe 2 d)	
	best finess, new filmess.
V. np. sandon. normal (0, 1, size d)	for i in range (n neix):
step: u) np. power cap. aps(v), 1/ bet)	if eardon randone () < pa:
redun sup.	meyer I ma yangan undan Claude how
Cold Allegan to 10 forest of	ness F. J. np. Handen unjon Clower bow
Out initialize-population (n. nest, n olin, lower-band,	traces 5.3 liter brown forest (17)
uppel-boud!	fines Tide fitnes-functions (nests lid)
eletur mp. eardon uniform (lower bound, upper be	
(n-nest, notion)	if (IKIation+1)-1,100=10 or Iklation = moralki
	print ( "Theater Fileration 1. 9, Best Fitness & Steel
def funes function (x):	return test nest, best fitness.
sutur np. sun(x ** 2)	
	Prest , 25
clef suchoo eramplos	n dim : 10
clef cuckoo seaut (m nest, m die, lower bound, upper	laver band5
orest 2 initialize population in maxites, po-0,25)	expu-bourd = 5.
orust 2 initialize population (n nest, n dim, lave but	
upper bout	The state of the s
1995	

### Code:

```
import numpy as np
import random
from scipy.special import gamma # Import gamma from scipy.special
# Levy flight function
def levy_flight(beta=1.5, d=1):
  sigma_u = np.power((gamma(1 + beta) * np.sin(np.pi * beta / 2) / gamma((1 + beta) / 2) * beta *
              np.cos(np.pi * beta / 2) ** 2), 1 / beta)
  u = np.random.normal(0, sigma_u, size=d)
  v = np.random.normal(0, 1, size=d)
  step = u / np.power(np.abs(v), 1 / beta)
  return step
# Initialize population (nests)
def initialize_population(n_nests, n_dim, lower_bound, upper_bound):
  return np.random.uniform(lower_bound, upper_bound, (n_nests, n_dim))
# Fitness function (Example: Sphere function)
def fitness_function(x):
  return np.sum(x ** 2)
# Cuckoo Search Algorithm
def cuckoo search(n nests, n dim, lower bound, upper bound, max iter, pa=0.25):
  # Step 1: Initialize nests randomly
  nests = initialize_population(n_nests, n_dim, lower_bound, upper_bound)
  # Step 2: Evaluate fitness of all nests
  fitness = np.array([fitness_function(nest) for nest in nests])
  # Track the best solution found so far
  best idx = np.argmin(fitness)
  best_nest = nests[best_idx]
  best_fitness = fitness[best_idx]
  # Start iterations
  for iteration in range(max_iter):
    # Generate new solutions using Levy flight
    for i in range(n nests):
       # Generate a new solution by Levy flight
       step = levy flight(d=n dim)
       new_nest = nests[i] + step * (nests[i] - best_nest)
       # Apply boundary conditions
       new nest = np.clip(new nest, lower bound, upper bound)
       # Evaluate the new solution
```

```
new_fitness = fitness_function(new_nest)
       # If the new solution is better, replace the old nest
       if new_fitness < fitness[i]:</pre>
         nests[i] = new nest
         fitness[i] = new_fitness
         # Update the best solution if necessary
         if new fitness < best fitness:
            best nest = new nest
            best_fitness = new_fitness
    # Abandon some of the worst nests and generate new random solutions
    for i in range(n_nests):
       if random.random() < pa: # with probability pa
          nests[i] = np.random.uniform(lower bound, upper bound, n dim)
         fitness[i] = fitness_function(nests[i])
    # Print progress every 100 iterations
    if (iteration + 1) % 100 == 0 or iteration == max iter - 1:
       print(f"Iteration {iteration + 1}, Best Fitness: {best_fitness}")
  return best_nest, best_fitness
# Parameters
n nests = 25 # Number of nests
n_dim = 10 # Dimensionality of the problem
lower bound = -5 # Lower bound for the search space
upper_bound = 5 # Upper bound for the search space
max iter = 1000 # Maximum number of iterations
pa = 0.25 # Probability of abandoning the worst nests
# Run Cuckoo Search
best_solution, best_solution_fitness = cuckoo_search(n_nests, n_dim, lower_bound, upper_bound,
max_iter, pa)
print("\nBest solution found: ", best_solution)
print("Best fitness value: ", best_solution_fitness)
```

Program 5 Grey Wolf Optimizer (GWO)

		CLASSMATE Date Page 11
	28/11/20	
		dab +: Greey way Openizes
	Tiboods?	
	114000	The grey world approved is a swarm intelligent
		individe Delicevision of the contract of
		a carrier a carrier of the
	Alexand .	a conige works with their collaboration
W	Call	hunting strategies
		loopert numer of ma
		Import mempy or mp
		Olef objective sunction (x):
	oping letter	Olf Objective Junction (X):
		XXx appr parts - avery agha
	Olhan	def grey welf-ophraizes (ob, func, din, sh, uh, wolves
2000		count, iterations):
(20)		alpha pos = np. zeros (olin)
	town	telta pos : np. zeros(din)
	TEN Elme	delsa pos, np. zelos (dim)
		alpho Sion. float (" 104")
13.	.01	sella scou float ( ing.)
	212010	della score, froat ("inf")
		completed agrituals and examine a second
155		wolves peritari. up. hander uniform (lb, ue, (wolves east)
,023/	Seal C. W.	dire))
		for literation in sampe (incations).
		for i in starge (wolves court):
		fitness, obj func (applies aprinvanti)
-		fitness , obj func (upowes postavnsti.T)  if fitness < alpha scou:
/	Lauras	alpha score, outpho pos. filmen, woolver- pon nouli7.
/	L	- south
/		elif fitness < beto_scot:
1		beta sione, beta por g fitness, coolves- positions sid capple

Date Page Let	classmate  Date Page 19:
	L roge
elle firmes & della siad:  cella siad, della per firmes, mon as partionis	date of the age of the
ally general della pos fitness, woolwas position	dimension : 5 dos and state of & dos
alle see from the see of the see	lower bound, -10
The state of the s	lower bound 10
in an Constant count):	
for i in stage (sooles - caust):	maxikiakin : 50
	A CONTRACTOR OF THE CONTRACTOR
0.0-9/1/2000	thest, position, test score gray work - approved C
911, 112, np. earders, rand (1), np. landon land	idhieldist wolld and
	supper bound, welves, maxiferation
1 A1 - 2 - a + 21 - 0	upper bound, welves, maniferation)
C1. 2+90	" But assition" pest all tron
No depthis himming (2)	Plust Best seen ;
Daspro, abs (C1 * alpho post) ] - wodies postroits	ay in (84) grassing (58) assort sugaration
VI alah antil at Dalaha	B Cerpout
X1. alpha-post; 7- As + Dalpha	& Desput
911, 20 rp, santa , are ,	Best poliker; [6.39416682 6.4005065 -6.595
A8, 2- a - 43- a	1.98919731 - 1.16874210e-00 7
91, 90, np. seandorn seard(), np. tondom souso  A 2, 2 a + 34 - a  C2, 2 42	Best Score: 2 4 025629534.
	Best Ston: 9-4-025629334.
D- beta, abs Cox beta par ( ) - woolver por humble )	A Co. N.
D-ben, ass (a' Deta-par (; 3- wolves per itural []) X2. Doto-par (; 1-A2 D) bute	UND STATE ( STATE STATE ) STATE OF " SOUTH - ATE
	and the same of th
Ill M. M. sanden, suand(), mp. suandern, rand()	def entral floor (44).
11, 12, np. random eard(), np. supratem tand()	441 1 10 rays (24, get 40 102).
C3 04 940	
C3, 9: 9(Q)	(1) white pay in whenever it would be gone
David as Cost and are an are headfilled	(L! Web 105) W wholey I am a
D-deta, aps(Cs + delta por [j] - woowes position []]  X3: delto_pos[j] - A3 * D-delta	dy get gentleger (sell 113):
X3: delto postij - A3 * D-delta	Transport 1
The state of the s	Allerent C. D. T.
ceoolues-perition[i][] (x1 + x2+x3)/3	
	of self-migranied is mouse
andred-positions [i] of clip ( worker portions [i], lb, 1)	Touch and a making
The state of the s	Chilly Tolk March 1919 22 Mar warmer
cutan alpha-gos, alpho nou	Diction 15 to 15 a - 25 . 25
I p p side	
The state of the s	

### Code:

```
import numpy as np
# Define the objective function to be minimized
def objective_function(x):
  return np.sum(x**2) # Example: Sphere function (minimization problem)
# Grey Wolf Optimizer (GWO) implementation
def grey_wolf_optimizer(obj_func, dim, lb, ub, wolves_count, iterations):
  # Initialize alpha, beta, and delta wolves' positions
  alpha pos = np.zeros(dim)
  beta_pos = np.zeros(dim)
  delta pos = np.zeros(dim)
  # Initialize alpha, beta, and delta wolves' fitness values
  alpha_score = float("inf") # Best fitness
  beta_score = float("inf") # Second-best fitness
  delta score = float("inf") # Third-best fitness
  # Initialize the positions of all wolves
  wolves positions = np.random.uniform(lb, ub, (wolves count, dim))
  # Main loop for optimization
  for iteration in range(iterations):
     for i in range(wolves count):
       # Calculate the fitness of the current wolf
       fitness = obj_func(wolves_positions[i])
       # Update alpha, beta, and delta wolves
       if fitness < alpha_score:
          alpha_score, alpha_pos = fitness, wolves_positions[i].copy()
       elif fitness < beta_score:
          beta_score, beta_pos = fitness, wolves_positions[i].copy()
       elif fitness < delta_score:
          delta score, delta pos = fitness, wolves positions[i].copy()
     # Update the positions of wolves
     for i in range(wolves count):
       for j in range(dim):
         # Coefficients
         a = 2 - 2 * (iteration / iterations) # Linearly decreases from 2 to 0
         r1, r2 = np.random.rand(), np.random.rand()
          A1 = 2 * a * r1 - a
         C1 = 2 * r2
```

```
D_{alpha} = abs(C1 * alpha_pos[i] - wolves_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] - wolves_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[i] - wolves_positions[i][i])
         X3 = delta_pos[j] - A3 * D_delta
         # Update position
         wolves positions[i][j] = (X1 + X2 + X3) / 3
       # Enforce bounds
       wolves_positions[i] = np.clip(wolves_positions[i], lb, ub)
  # Return the best solution
  return alpha_pos, alpha_score
# Parameters
dimension = 5 # Number of variables
lower_bound = -10 # Lower bound of variables
upper_bound = 10 # Upper bound of variables
wolves = 30
                # Number of wolves in the pack
max iterations = 50 # Maximum number of iterations
# Run the GWO algorithm
best_position, best_score = grey_wolf_optimizer(
  objective_function, dimension, lower_bound, upper_bound, wolves, max_iterations
)
print("Best position:", best_position)
print("Best score:", best_score)
```

Program 6
Parallel Cellular Algorithms and Programs

onia	
Page 20	classmate
	Dote Page 01
19/12/29 Parallel Cellular Algorithm.	19/19/20
Joh W. Furth	supate states (self):
and collistal algorithms are enspired by	new grid, self , gud copy ()
Harallel accion of prological cells that come	for 1 in sange (suy great size EOI):
the function pougle and distributed manner	for jun stang (self. grid. 31cc Eis):
Pavallel cellulal algorithms are vispited by the functioning of toological cells that expense on a highly pavaller and distributed manning	neighbors = self get_neighbors (i))
import numpy as np.	sef. gad, new gud
import fluide (inchor (il):	Production .
dy gastriga function (2):	run (zulf);
901 xi len(x) + sum((xi **2 - B) * pp. cos (2 * pp. xi)  for xi inx)	for Helation in range (self-1412 Kons)?
seekin H which for xi iox	self evaluate (thress ()
a same averibm'	self. update, states ()
class Parallel Cellular Algorithm.	Phurit (f" Decapor Silvation + 13/3llfiterature 9")
af ind ( Set, grid-size, (5,5), die 2, neighborhood, moon!, jewoon :100)?	2 00 C 100 1/2 + 2 - 4 + 100 C 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
	pount (" Poest solution: Eself. Dest solution y")
self, grid-size = grid-size	pount (" Best solution: Iself. Dest, solution y")
self-dein, den	presid (f" Best Fitnes: { sety best fitnes: 6 f3")
self neighborhood, neigh borhood	The interfeet, agent we do good to to the color to the
	if - name " main.
self. g.id, np. evandom. uniform bounds [0], towall] grid stelli], din)	poa, Parallel Celleratigourano Igreio. sire, (10,10),
grid stell, din	din 1, neighborhoods 'moon')
	pca. xunl)
gelf. filmon, np. zeros ((grid, size [0], grid, size [1]))	The second secon
	Allout Was was said was now the
def evalual filmes (self):	Output.
for i in range (sext. grad, erre (OD):	seeking of seeking with seek parantal seek paratal
(or; un large (sef. good sire (i):	Optionization Complete
ref. fineulij]. eastrigin for (sef. garp; j])	Best Solection: [-0. 0337945 -0.0340274]
	Best Pitness, D. 451717,
11 and hharde 41 1 2	STAND STALKED JOHN SOUTH
dy get neighbors (self, i, j):	
neighbou,[]	at splethin (sell)
direction. []	Lave framerican
if self. neighborhood 2. mooce!	177 - 140 - 170
ellit 80H; marting	The word (set the sut):
ellif set. neinbanara 2 > 'von neuman':	super color randor superest proper success.
directions: [(-1,0),(0,-1)(0,1)]-	sukun selekur.
	AUNI EMA

## Code: import numpy as np def rastrigin function(x): Rastrigin function for optimization. It has a global minimum at x = 0. $f(x) = 10n + sum(x_i^2 - 10 * cos(2 * pi * x_i))$ A = 10return A \* len(x) + sum((xi \*\* 2 - A \* np.cos(2 \* np.pi \* xi))) for xi in x)class ParallelCellularAlgorithm: def \_\_init\_\_(self, grid\_size=(5, 5), dim=2, neighborhood='moore', iterations=100, bounds=(-5.12, 5.12)): Initialize the parameters for the cellular algorithm. :param grid\_size: Tuple defining the dimensions of the grid (rows, columns) :param dim: Dimensionality of the solution space :param neighborhood: Type of neighborhood ('moore' or 'von\_neumann') :param iterations: Number of iterations to run :param bounds: Lower and upper bounds for the solution space self.grid size = grid size self.dim = dimself.neighborhood = neighborhood self.iterations = iterations self.bounds = bounds# Initialize the grid with random solutions self.grid = np.random.uniform(bounds[0], bounds[1], size=(grid\_size[0], grid\_size[1], dim)) self.fitness = np.zeros((grid\_size[0], grid\_size[1])) # Store the best solution and its fitness self.best solution = None self.best fitness = float('inf') def evaluate fitness(self): Evaluate fitness of each cell in the grid. for i in range(self.grid size[0]): for j in range(self.grid\_size[1]): self.fitness[i, j] = rastrigin\_function(self.grid[i, j]) # Update the best solution if self.fitness[i, j] < self.best\_fitness:

```
self.best fitness = self.fitness[i, j]
          self.best_solution = self.grid[i, i].copy()
def get_neighbors(self, i, j):
  Get neighboring cells based on the chosen neighborhood type.
  :param i: Row index of the current cell
  :param j: Column index of the current cell
  :return: List of neighboring cells
  neighbors = []
  directions = []
  if self.neighborhood == 'moore':
     # Moore neighborhood: 8 neighbors (including diagonals)
     directions = [(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1), (1, -1), (1, 0), (1, 1)]
  elif self.neighborhood == 'von_neumann':
     # Von Neumann neighborhood: 4 neighbors (up, down, left, right)
     directions = [(-1, 0), (0, -1), (0, 1), (1, 0)]
  for di, di in directions:
     ni, nj = i + di, j + dj
     if 0 \le ni \le self.grid\_size[0] and 0 \le nj \le self.grid\_size[1]:
       neighbors.append(self.grid[ni, nj])
  return neighbors
def update_states(self):
  Update the state of each cell based on its neighbors.
  new_grid = self.grid.copy()
  for i in range(self.grid_size[0]):
     for j in range(self.grid_size[1]):
        neighbors = self.get_neighbors(i, j)
       # Calculate the average position of neighbors
        avg_neighbor = np.mean(neighbors, axis=0)
       # Move the current cell slightly towards the average neighbor position
        new_grid[i, j] = self.grid[i, j] + 0.1 * (avg_neighbor - self.grid[i, j])
       # Ensure the new solution stays within bounds
       new_grid[i, j] = np.clip(new_grid[i, j], self.bounds[0], self.bounds[1])
  self.grid = new_grid
def run(self):
```

```
Run the Parallel Cellular Algorithm.

"""

for iteration in range(self.iterations):

# Step 1: Evaluate fitness of the current grid
self.evaluate_fitness()

# Step 2: Update the states of all cells
self.update_states()

# Print progress
print(f"Iteration {iteration + 1}/{self.iterations}, Best Fitness: {self.best_fitness:.6f}")

print("Optimization complete.")
print(f"Best Solution: {self.best_solution}")
print(f"Best Fitness: {self.best_fitness:.6f}")

if __name__ == "__main__":

# Initialize and run the algorithm
pca = ParallelCellularAlgorithm(grid_size=(10, 10), dim=2, neighborhood='moore', iterations=50)
pca.run()
```

**Program 7**Optimization via Gene Expression Algorithms

Page 2R	classmite
	Date Page 23.
dat 9: Optimization via Gone Engenession Algoritan	
dab 9: Uphimizano	alf mutale (3ett, genl-sequence):
2: OUR UNSWINE THE STOLE	for I in sand (len (gen. segun a)):
Core Expression in living again	if elandern grandom () < gelf mutation grat:
nances at translation of according	geni seque u [i] + 2 np. sandom uniform (-1.0, 1.0)
The propose circulate DNP into functional	gene eguene (i) : np. dip (gene leguenti)
The Propose cinveles at TNP into functional encoded as TNP into functional	veturn gene Reguena. Ref. bouros (:2)
persters.	
	defe gave expansion self, gene squaree):
import numpy as np	sulun gene seguene.
imper Gundem	0.412
impart manarm def martingm-function (x):	ay sum (sety):  101 anetakon in range (sety generation):
A: 10 (1): 1 = 2 - A = DD: CO(2)	for generation in sample (self-generation):
sietem 4.0 m(x) + sim((xi = -2 - 4 - 10 cos(2 - 104)	litnes values , (sex warms)
XIII go XI (nx)	
( ) who had their solution that I have	best-idx, np. argmin(fitness.value)
class Gene Expression Algorithm:  class Gene Expression Algorithm:  (00 4 000 510 , 100, gene 1:00, tounds (-5.12,5:11)).	selected pop? Self. selenone
class Gene Expression Higgs and 100, gene 100, tounds (-5.2,5:11).	next_population > []
self pop sire , pop sire	for 1 is marge (0, self: pop. size, 2):
self-fighe sta for length	parent 1. select-popsis
self gen leyth, gen leyth	pount 2, select, pop [i+1]
self mutation rate, mudation, rate	offsprise, offsprise, self crossover (p1, p2)
sey bourds , bounds.	next pop append (sey mutate (off & put 1,))
	acel acceletant next population
all eardon gene sequence (sey):  seekun np. soundon uniform (sey bourds (0), sey band (),  sire, self. eppl. legth)	Self. population: next population
seetun . pp. grandom. unt own (sex bourds to), sex bando ()	( * O - · · · · · · · · · · · · )
sire. self. gene. legth)	print ( Optimization complete)
	printt ( f "Best SOI) { self. Best SOI?"
	Print ("f" Best Fitnes: [ self. Dest firms: 6/3.
def evaluate fitness ( set, gens - seguna):	
gutum reastling function (gent equell)	gla. sun().
10	7
ef Selection (Self):	210 0 1 12 2141
	Opp Optimization Complete -0.01908(02)
foundment size 3	Best Sob: [-0.02796 851 -0.01908597]
Selected : []	Best Sob: [-0.02796 851 -0.01908597]  Best Sob: [-0.02796 851 -0.01908597]  Best Firmer: 0.226970.
601_ In rays (304. pop, sire):	
townament , random. Sample (sey, population, towname) siu)	
siu)	
seethin Selected.	

```
Code:
import numpy as np
import random
def rastrigin function(x):
  Rastrigin function for optimization. It has a global minimum at x = 0.
  f(x) = 10n + sum(x i^2 - 10 * cos(2 * pi * x i))
  A = 10
  return A * len(x) + sum((xi ** 2 - A * np.cos(2 * np.pi * xi))) for xi in x)
class GeneExpressionAlgorithm:
  def __init__(self, population_size=100, gene_length=10, mutation_rate=0.01, crossover_rate=0.7,
generations=100, bounds=(-5.12, 5.12)):
    Initialize the parameters for the Gene Expression Algorithm.
    :param population_size: Number of genetic sequences in the population
     :param gene length: Length of each genetic sequence
    :param mutation_rate: Probability of mutation
    :param crossover rate: Probability of crossover
     :param generations: Number of generations to evolve
    :param bounds: Lower and upper bounds for the solution space
    self.population_size = population_size
    self.gene length = gene length
    self.mutation_rate = mutation_rate
    self.crossover_rate = crossover_rate
    self.generations = generations
    self.bounds = bounds
    # Initialize population with random solutions
    self.population = [self.random_gene_sequence() for _ in range(population_size)]
    self.best solution = None
    self.best_fitness = float('inf')
  def random_gene_sequence(self):
     """Generate a random gene sequence within bounds."""
    return np.random.uniform(self.bounds[0], self.bounds[1], size=self.gene_length)
  def evaluate fitness(self, gene sequence):
     """Evaluate the fitness of a gene sequence using the Rastrigin function."""
    return rastrigin_function(gene_sequence)
  def selection(self):
     """Select individuals for reproduction using tournament selection."""
    tournament\_size = 3
    selected = []
   <sup>2</sup>∮or _ in range(self.population_size):
       tournament = random.sample(self.population, tournament_size)
       tournament_fitness = [self.evaluate_fitness(ind) for ind in tournament]
```

```
winner = tournament[np.argmin(tournament fitness)]
     selected.append(winner)
  return selected
def crossover(self, parent1, parent2):
  """Perform crossover between two parents to produce offspring."""
  if random.random() < self.crossover_rate:</pre>
     point = random.randint(1, self.gene_length - 1)
     offspring1 = np.concatenate((parent1[:point], parent2[point:]))
     offspring2 = np.concatenate((parent2[:point], parent1[point:]))
     return offspring1, offspring2
  return parent1, parent2
def mutate(self, gene_sequence):
  """Apply mutation to a gene sequence."""
  for i in range(len(gene_sequence)):
     if random.random() < self.mutation rate:
       gene_sequence[i] += np.random.uniform(-1.0, 1.0)
       gene_sequence[i] = np.clip(gene_sequence[i], self.bounds[0], self.bounds[1])
  return gene sequence
def gene expression(self, gene sequence):
  """Translate a genetic sequence into a functional solution (no changes needed here)."""
  return gene_sequence
def run(self):
  """Run the Gene Expression Algorithm."""
  for generation in range(self.generations):
     # Step 1: Evaluate fitness and find the best solution
     fitness_values = [self.evaluate_fitness(ind) for ind in self.population]
     best idx = np.argmin(fitness values)
     if fitness values[best idx] < self.best fitness:
       self.best fitness = fitness values[best idx]
       self.best_solution = self.population[best_idx].copy()
     # Step 2: Selection
     selected_population = self.selection()
     # Step 3: Crossover and Mutation
     next_population = []
     for i in range(0, self.population_size, 2):
       parent1 = selected population[i]
       parent2 = selected\_population[i + 1]
       offspring1, offspring2 = self.crossover(parent1, parent2)
       next_population.append(self.mutate(offspring1))
       next_population.append(self.mutate(offspring2))
     self.population = next_population
```

```
print("Optimization complete.")
    print(f"Best Solution: {self.best_solution}")
    print(f"Best Fitness: {self.best_fitness:.6f}")

if __name__ == "__main__":
    # Initialize and run the Gene Expression Algorithm
    gea = GeneExpressionAlgorithm(population_size=50, gene_length=2, generations=50)
    gea.run()
```