VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.



LAB RECORD

Bio Inspired Systems (23CS5BSBIS)

Submitted by

SUJAY PRASAD P V(1BM23CS422)

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 **Sujay Prasad P V (1BM23CS422)**, 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.

Syed Akram	Dr. Kavitha Sooda
Assistant Professor	Professor & HOD
Department of CSE, BMSCE	Department of CSE, BMSCE

Index

Sl. No.	Date	Experiment Title	Page No.
1	03/10/24	GENETIC ALGORITHM	4-9
2	24/10/24	PARTICLE SWARM OPTIMIZATION	10-14
3	07/11/24	ANT COLONY OPTIMIZATION	15-21
4	21/11/24	CUCKOO SEARCH ALGORITHM	22-24
5	28/11/24	GREY WOLF OPTIMIZATION	25-28
6	12/12/24	PARALLEL CELLULAR ALGORITHM	29-33
7	12/12/24	GENE EXPRESSION ALGORITHM	34-38

Github Link:

https://github.com/SujayPrasadPV/BIS/tree/main

Program 1

Genetic Algorithm for Optimization Problems

```
Cenelic algorithm
import random
  Population size = 100
  erenes = "abcdefghigkImnoparsturwxyz ABCDEFEHITS KLMUO
   PQRSTUVWXYZ 123456789; -: :- !"116/11=1@$467}"
   Target ="smile!"
  class individual Cobject):
         def -init - (self, chromosome)
               Self. chomosome = chromosome
               self. fitness = self. cal-fines(1)
         Coasi method
         det mutated gones (salt):
                 gene = random. choice (evenel)
                 return gone
         & dann thad
            det create grome (self).
                   grown_len = len Tangel)
                    redum [self. multideg-gores () for in songe (groweld)
           det mate (self, pour 2).
                  Child-chromopone = ED
                  for get, gp2 in zip (self. choomojome, pare, dware
                       prob = random randoms
                        if pro 6 < 0.45:
                             child - chronosome pare-chromosom
                         elif prob < 0.90.
                            child - shromosome appoind (gpe)
```

Particle Swarm Optimization import random "mport math import Copy tradui sys Pitness_rastrigin (position): return sum ((xi*xi) -(10" math.cos (2" math.pi. *xi))+lo to xi in position) det litress - sphere (position) return sum (xi+xi for xi in position) chass particle: det __init_- (self, fatness, dim, minoc, maxs, seed): self. and = random. Random (seed) self. position- [(mana-mina) * self. rnd randomo + minx for - in range (dim) fell relocity = [cmarce ming sell and randomes + minor for in range (dim) self. best -part-pol = self. position() Sett. Fitness = 4 thress (self position) self. best-part- fitness val = self. fitness det psolfitars, max-ilor, n, dim, mino, manex): W, ZI, (2= 0.729, 1.49 HHS, 1.49445 rand - random. Rundom 100 Swarm = [Partide (fitness, dim, minx, manx) (5) for in range [n]] best - swarm-pos, best - swarm- Afthessval = [0.0] +dim,

sysp. flock-intomax

```
Code:
import random
POPULATION_SIZE = 100
GENES = "'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOP
QRSTUVWXYZ 1234567890, .-;:_!"#%&/()=?@${[]}""
TARGET = "I love GeeksforGeeks"
class Individual(object):
      Class representing individual in population
      def __init__(self, chromosome):
             self.chromosome = chromosome
             self.fitness = self.cal fitness()
       @classmethod
       def mutated_genes(self):
             create random genes for mutation
             global GENES
             gene = random.choice(GENES)
             return gene
       @classmethod
      def create_gnome(self):
             create chromosome or string of genes
             global TARGET
             gnome len = len(TARGET)
             return [self.mutated_genes() for _ in range(gnome_len)]
      def mate(self, par2):
             Perform mating and produce new offspring
             child chromosome = []
             for gp1, gp2 in zip(self.chromosome, par2.chromosome):
                    prob = random.random()
                    if prob < 0.45:
```

```
child_chromosome.append(gp1)
                     elif prob < 0.90:
                            child_chromosome.append(gp2)
                     else:
                            child_chromosome.append(self.mutated_genes())
              return Individual(child_chromosome)
       def cal_fitness(self):
              Calculate fitness score, it is the number of
              characters in string which differ from target
              string.
              global TARGET
              fitness = 0
              for gs, gt in zip(self.chromosome, TARGET):
                     if gs != gt: fitness+= 1
              return fitness
def main():
       global POPULATION_SIZE
       generation = 1
       found = False
       population = []
       for _ in range(POPULATION_SIZE):
                            gnome = Individual.create_gnome()
                            population.append(Individual(gnome))
       while not found:
              population = sorted(population, key = lambda x:x.fitness)
              if population[0].fitness <= 0:
                     found = True
                     break
              new_generation = []
              s = int((10*POPULATION_SIZE)/100)
              new_generation.extend(population[:s])
```

```
s = int((90*POPULATION_SIZE)/100)
              for _ in range(s):
                     parent1 = random.choice(population[:50])
                     parent2 = random.choice(population[:50])
                     child = parent1.mate(parent2)
                     new_generation.append(child)
              population = new_generation
              print("Generation: {}\tString: {}\tFitness: {}".
                      format(generation,
                      "".join(population[0].chromosome),
                     population[0].fitness))
              generation += 1
       print("Generation: {}\tString: {}\tFitness: {}".
              format(generation,
              "".join(population[0].chromosome),
              population[0].fitness))
if __name__ == '__main__':
       main()
```

Program 2

PARTICLE SWARM OPTIMIZATION

Algorithm: Particle Swarm Optimization import random "mport math import Copy + radini sys det fitness_rastrigin (position): return sum ((xi*xi) -(10 math.cos (2 math.pi. *x:))+10 to xi in portion) def fitness - sphere (position) return sum (ri+zi for xi in position) chass particle: det __ init_ - (self, fatness, dim, minoc, maxs, seed). self. and = random. Random (seed) self. position- [(mana-mina) + self. and randoms) + minx for - in range (dim) self relocity = [comare ming sell and randoms + minse for in range (dim) self. best -part-pol = self. position (:) Sett. Fitness = titness (self. position) self. best-part- fitness val = self. fitness det psolfitness, max-ilor, n, dim, mino, masex): W, TI, (2= 0.729, 1.49 HHS, 1.494MS rand - random. Rundom (0) Swarm = [Partide (fitness idim, minx, manx si) (3) tor in range [n]] best - swarm-pos, best-swarm- Ifthessved = [0.0] + dim, syst. flock - into max

```
for pin swarm
               if p. Athers 2 best - swarm. fitnessed:
                    best warm throwal = p. fitness
                     best swarm-pos = p. parition []
        for Iter in range (maxiter).
              if Iter 1- (0==0 and I for >01
                     print (f" Iter = \ Iter > best fitness = { best-swarmalide
               for p in swarm:
                     for kin range (diren):
                           ri, re = rand random (), rad random ()
                            p. velocity[w] = w* p. velocity (x)+c1* r1* ( p. best
                                             part-pos(E) - p.portion(E)) +(2*
                                             Y2* (best swarm-pos[k]-parting
                              p. velocity [1] = max(min(p. velocity (12), max) min)
                     p. position = [p. position (F) + p. velocity (F) for le invangeld
                     p. Bost trans - fitness (provision)
                    if p. fitness & p. best - pourt - fit nessual:
                             p. best-part-tinesnal= p.f. itness
                               P. beck-part-pol = p.polition []
                     + p.fiteren < pober + swar m_ fitner val:
                            belt - swarm-litress ud = pfitnell
                               best - swarn - pos = p. position []
     return bist-swarm-pos
det run pso lithes, dim, minx, maxx):
             print ( +" enach is to minimize the tendion in faim be eviable
              print (1" function has known min = 0.0 at(1; ; join(10) to int
                       ("(0
               numepurlides, may- "ter = 50,100.
```

but position = pro(fil nell , mareiter, nun-particles dum

For so in bill-polition found of isomilification for so in bill-polition 1)

print [f" titues of best column = (fitness (bost - position): 61 pin ")

print ("In agin Pso to. Rollingin function in")
run - 150 (fitness -radrigin, 3, -10.0, co.0)

print ("In Begin Pso for Sphere fundion In")
run-pso (tithous sphere , 3, -10, 0, 10.0)

output:

Belt solution found:

[50000000, 1,00000001, 10000000]

fitures of best solution = 0.000000

```
Code:
import random
import math
import copy
import sys
def fitness_rastrigin(position):
  return sum((xi * xi) - (10 * math.cos(2 * math.pi * xi)) + 10 for xi in position)
def fitness_sphere(position):
  return sum(xi * xi for xi in position)
class Particle:
  def init (self, fitness, dim, minx, maxx, seed):
    self.rnd = random.Random(seed)
    self.position = [(maxx - minx) * self.rnd.random() + minx for _ in range(dim)]
    self.velocity = [(maxx - minx) * self.rnd.random() + minx for _ in range(dim)]
    self.best_part_pos = self.position[:]
    self.fitness = fitness(self.position)
    self.best_part_fitnessVal = self.fitness
def pso(fitness, max_iter, n, dim, minx, maxx):
  w, c1, c2 = 0.729, 1.49445, 1.49445
  rnd = random.Random(0)
  swarm = [Particle(fitness, dim, minx, maxx, i) for i in range(n)]
  best swarm pos, best swarm fitnessVal = [0.0] * dim, sys.float info.max
  for p in swarm:
    if p.fitness < best swarm fitness Val:
      best_swarm_fitnessVal = p.fitness
       best_swarm_pos = p.position[:]
  for Iter in range(max iter):
    if Iter % 10 == 0 and Iter > 1:
       print(f"Iter = {Iter} best fitness = {best_swarm_fitnessVal:.3f}")
    for p in swarm:
       for k in range(dim):
         r1, r2 = rnd.random(), rnd.random()
         (best_swarm_pos[k] - p.position[k])
         p.velocity[k] = max(min(p.velocity[k], maxx), minx)
       p.position = [p.position[k] + p.velocity[k] for k in range(dim)]
       p.fitness = fitness(p.position)
      if p.fitness < p.best_part_fitnessVal:
```

```
p.best_part_fitnessVal = p.fitness
          p.best_part_pos = p.position[:]
       if p.fitness < best_swarm_fitnessVal:
          best_swarm_fitnessVal = p.fitness
          best_swarm_pos = p.position[:]
  return best_swarm_pos
def run_pso(fitness, dim, minx, maxx):
  print(f"Goal is to minimize the function in {dim} variables")
  print(f"Function has known min = 0.0 at ({', '.join(['0'] * (dim - 1))}, 0)")
  num_particles, max_iter = 50, 100
  best_position = pso(fitness, max_iter, num_particles, dim, minx, maxx)
  print(f"Best solution found: {', '.join([f'{x:.6f}' for x in best_position])}")
  print(f"Fitness of best solution = {fitness(best_position):.6f}\n")
print("\nBegin PSO for Rastrigin function\n")
run_pso(fitness_rastrigin, 3, -10.0, 10.0)
print("\nBegin PSO for Sphere function\n")
run_pso(fitness_sphere, 3, -10.0, 10.0)
```

ANT COLONY OPTIMIZATION

```
Ant colony Optimization
import random
 anna former traini
  "unport math
  class city:
          det - init - (sel, x, y):
                  Sele. x = x
                  self. 4 = x
          det d'home (rell, c'ty):
                 return math. copt ((self.x-city.x) = 2 + (self.y-city.y) = 2)
das1 A(0-75P:
     def -- init -- (self, cities, hum-ants, nun-iterations, alpha=10, bela=20,
                     sho=0.5, a =09):
           sat cites = cities
           self. nun-aut = mun- eunts
           rat. hun_iterations = nun_iterations
            set-alpha = alpha
            self. beta = beta
             call . rho = rho
             sal-an = an
             set noncities = lenterities)
             self, pheromone = np. onestilet. numerities, self. numerities)
             self. Even it ic Emp. zeros (Helf, num-citie), self. numcitie)
              get. best-tour = warp
              self. best-tour length = float(' 7nf)
              for in range (rell numcitics):
                      for sin range [ :+ 1, sept. numcities]:
                            (City = citical ED. dillance (Citical ET)
                            Sat. Lewitic CiJCjJ=1.0 (dist i (dist)
                             set. hearlitic [3][1] = self. heartific [3]
```

def scheet next city (sele, current city, visited-cities). probabilities = upzeros(salt, municities). total - phenomen = 00 for casy in range (self. nun-cities): it city not in writed-cities: theromore = sel. phronore (curact ety) (c #4) * self alpha Marvitic = red. neurolitic Ecumations [city] self beta probabilities [Ely] = pheromore + heard Fie total - phromonet = probabilities (city) 1 t total-phenomone == 0: return random thoise ((city forcity is range @ (self. name cities if city notify vitited cities) oroloods Fiel 1= total -pheromone if random random () 2 self-qu: next - city = np. argmex (protabilitic)) elst! next-city inproadon. Noice (nell - humallies -pr= probabilisties) return next-city def supdute-pheromone (celf, anti): self. phromane 4 = (1-self, who) for aut in auts: phenomone deposit = 1 of anti-tour length for i in range (self. nun-cities): current-city = and tour [] next-city = and tour [() + 1) - (elf non citie) s cit. promone [current city] [next city] + promonenes

det run (self) for iteration in range (12) num-iterational ants = [Ant (self. num_cities, self) for in rangely for " in range (sell num citic): (whent city - and tour (1) next city = ant town (Cit) : with num city Self phonomone Ecu for and in out : if out tour len ant. Construct - rolldian() sel. epdate_pheromone(contr) for and mant! it and tour-length & self. both tour langth Seef best tour - loughth = and four - lough gelf. belt-tour =ant. tour print (f" I tandion ? 'tandios + 1) / self-run itali Be A Tour langth = {(ell. best bour -langth} a) return felt. best-tour, self. best-tour-lugth class Ant: det -- init -- (self, numeries, acostp): Self. nuncities = muncities sof acoup = aco - Hp Set Hour = [] self. Lour_length = 0.0 Let construct-solution (ICLF): Stort City = random - rand int (0, cell new = celtiel - 1) self tour = [ctart-city] set villed - (itil) = set(sell . tour) Current City = Hast City While I an (self. four) & self . nam_cifiess next-city=sell_aco-tip. reled-next-zity (une

- city, visited cities)

16 | Page

sef . tour append (next ecety) Virited with a add (next-city) -Self. tour - rough + +a = self. a co-tip. cities [current_ Obj]. distance (eds. aco_tsp. cities Front-city) (unat - city = rox 1 - l'ity self tour length += sel aco-tp. cities Belf tour E-T) d war ((elf. a co-tip. (itil [self. tour (0)]) ! f_name -- == "-- main -- ". c'ities = (city (0,0), city(1,3), city (4,7), city (6,1), L'ity (3,0) alo = ala TIP (ities = cittes, normant zog numiteration)= con alpha = (00, beta = 20, rho=0.5, NO=09) best-tour, text-tour-laught = aco runs privat 1"In Best ton found: ", best town) put (" In Best tour length:", best tour length output: Belt to us town 2: [3, Np. int 64(2), np: int 64(1), np: int 64(0) NP 'm+647 4D to 1 tops light : 19. 152982

```
Code:
import random
import numpy as np
import math
class City:
  def \underline{\quad} init\underline{\quad} (self, x, y):
     self.x = x
     self.y = y
  def distance(self, city):
     return math.sqrt((self.x - city.x)**2 + (self.y - city.y)**2)
class ACO_TSP:
  def __init__(self, cities, num_ants, num_iterations, alpha=1.0, beta=2.0, rho=0.5,
q0=0.9):
     self.cities = cities
     self.num ants = num ants
     self.num iterations = num iterations
     self.alpha = alpha
     self.beta = beta
     self.rho = rho
     self.q0 = q0
     self.num_cities = len(cities)
     self.pheromone = np.ones((self.num_cities, self.num_cities))
     self.heuristic = np.zeros((self.num_cities, self.num_cities)) (inverse of distance)
     self.best tour = None
     self.best_tour_length = float('inf')
     for i in range(self.num cities):
       for j in range(i + 1, self.num_cities):
          dist = cities[i].distance(cities[i])
          self.heuristic[i][j] = 1.0 / dist if dist != 0 else 0
          self.heuristic[i][i] = self.heuristic[i][i]
  def select_next_city(self, current_city, visited_cities):
     probabilities = np.zeros(self.num_cities)
     total_pheromone = 0.0
```

```
for city in range(self.num_cities):
       if city not in visited_cities:
         pheromone = self.pheromone[current_city][city] ** self.alpha
         heuristic = self.heuristic[current_city][city] ** self.beta
         probabilities[city] = pheromone * heuristic
         total pheromone += probabilities[city]
    if total_pheromone == 0:
       return random.choice([city for city in range(self.num_cities) if city not in
visited_cities])
    probabilities /= total_pheromone
    if random.random() < self.q0:
       next_city = np.argmax(probabilities)
    else:
       next_city = np.random.choice(self.num_cities, p=probabilities)
    return next_city
  def update pheromone(self, ants):
    self.pheromone *= (1 - self.rho)
    for ant in ants:
       pheromone_deposit = 1.0 / ant.tour_length
       for i in range(self.num_cities):
         current_city = ant.tour[i]
         next\_city = ant.tour[(i + 1) \% self.num\_cities]
         self.pheromone[current_city][next_city] += pheromone_deposit
         self.pheromone[next_city][current_city] += pheromone_deposit
  def run(self):
    for iteration in range(self.num_iterations):
       ants = [Ant(self.num_cities, self) for _ in range(self.num_ants)]
       for ant in ants:
          ant.construct_solution()
       self.update_pheromone(ants)
```

```
for ant in ants:
          if ant.tour_length < self.best_tour_length:</pre>
             self.best_tour_length = ant.tour_length
             self.best tour = ant.tour
       print(f''Iteration \{iteration + 1\}/\{self.num iterations\}: Best Tour Length =
{self.best_tour_length}")
     return self.best_tour, self.best_tour_length
class Ant:
  def __init__(self, num_cities, aco_tsp):
     self.num_cities = num_cities
     self.aco_tsp = aco_tsp
     self.tour = []
     self.tour length = 0.0
  def construct solution(self):
     start city = random.randint(0, self.num cities - 1)
     self.tour = [start city]
     self.tour length = 0.0
     visited_cities = set(self.tour)
     current_city = start_city
     while len(self.tour) < self.num_cities:
       next_city = self.aco_tsp.select_next_city(current_city, visited_cities)
       self.tour.append(next_city)
       visited_cities.add(next_city)
       self.tour_length +=
self.aco_tsp.cities[current_city].distance(self.aco_tsp.cities[next_city])
       current_city = next_city
     self.tour length += self.aco tsp.cities[self.tour[-
1]].distance(self.aco tsp.cities[self.tour[0]])
if __name__ == "__main__":
  cities = [City(0, 0), City(1, 3), City(4, 3), City(6, 1), City(3, 0)]
  aco = ACO_TSP(cities=cities, num_ants=10, num_iterations=100, alpha=1.0,
beta=2.0, rho=0.5, q0=0.9)
  best_tour, best_tour_length = aco.run()
  print("\nBest tour found:", best_tour)print("Best tour length:", best_tour_length)
```

CUCKOO SEARCH ALGORITHM

```
Cuckoo Search
     "import numpy as no o
      from scipy special import gamma
      det objective x):
           return np. sum (x * 2)
     def levy flight (bela, dim):
          Sigma = (gamma(1+bela)*np.sin (np.pi*bela))/(gamma(1+bel
                   bela* np. power (2, (bela-1)/2)) + (1/6ela)
          u = np random normal (0, sigma, dim)
           v= np. random. normal (0,1,dim)
           return u/ np. abs (v) ** (+/bela)
  del cuckoo_search(ob; func (next) for next in nexts])
        neste = np. random uniform (bounde [o], bounde [], (n, dim))
        Pitness = np. array ( [ bbj = func ( nest ) for nest in nests])
         best-nest = nests Enp. argmin (titness)
         best-litness = np. min (fitness)
         for - in range (max-iter):
new-necti = np.copy (nests)
                for : in range (N):
                     step = leng - flight (15, dim)
                     quer los + Fill the = Fill + 1901 - step
                      new-nette [:] = np. clip (new nette [:], bound ([])
               new -fitner = np. agray ( Eob; - func ( next) for next in how me
               for: in range (N):
                   in p. random- sand > > pa and new-fitners EJZ fitners
                         nestici) = new - nisti (i)
                         Cilyantif-war = Cilliantif
              best_nect-idx = np. argmin (4thneck)
               beil-nest = negli [beit_not-i as]
               best_ titues = tituess [best-ned-ido)
    return best-nest best-sitness
```

bound(= [-5, 3) best-nest, buil-fitness = cuckoo-rearch (objects produm, bounds) bejorg (to Best West. (port west).) print (f" Bect titral: (bet-titneschi) Output: Best Nest: [-2.7169, 0.5995 1.1431 -0.7824 -0.3176 -0.2455 1.5899 24114 -4.0075 13881) Best Fitness: 36.14801947

```
Code:
```

```
import numpy as np
from scipy.special import gamma
def objective(x):
  return np.sum(x^{**}2)
def levy flight(beta, dim):
  sigma = (gamma(1+beta)*np.sin(np.pi*beta/2) /
        (gamma((1+beta)/2)*beta*np.power(2, (beta-1)/2)))**(1/beta)
  u = np.random.normal(0, sigma, dim)
  v = np.random.normal(0, 1, dim)
  return u / np.abs(v)**(1/beta)
def cuckoo_search(obj_func, dim, bounds, N=20, pa=0.25, max_iter=100):
  nests = np.random.uniform(bounds[0], bounds[1], (N, dim))
  fitness = np.array([obj_func(nest) for nest in nests])
  best nest = nests[np.argmin(fitness)]
  best fitness = np.min(fitness)
  for in range(max iter):
    new_nests = np.copy(nests)
    for i in range(N):
       step = levy_flight(1.5, dim) # Lévy exponent 1.5
       new_nests[i] = nests[i] + 0.01 * step
       new_nests[i] = np.clip(new_nests[i], bounds[0], bounds[1])
    new_fitness = np.array([obj_func(nest) for nest in new_nests])
    for i in range(N):
       if np.random.rand() < pa and new_fitness[i] < fitness[i]:
         nests[i] = new nests[i]
         fitness[i] = new_fitness[i]
    best nest idx = np.argmin(fitness)
    best_nest = nests[best_nest_idx]
    best fitness = fitness[best nest idx]
  return best nest, best fitness
dim = 10
bounds = [-5, 5]
best_nest, best_fitness = cuckoo_search(objective, dim, bounds)
print(f"Best Nest: {best_nest}")
print(f"Best Fitness: {best_fitness}")
```

GREY WOLF OPTIMIZATION

```
import number on ub
del quo (ob; -fundion, d'in, cearch - agents, max iter, 16, ub).
      Alpha = pos = npzeros(dim)
      Beta por = np.zeros (dim)
       peltapor = np. zeros ( d'in)
        Alpha-score = flood("inf")
        Beta - (core = float ("inf")
        Delta-source = float (", ")
        positions = representation without (16, 46, (search agents dim))
        for iteration in range (maxiter):
             for; in range (search agents)
                    (du, d) [i3 moiting) quil. qn = [i] moiting
                     fitness = ob; -tundion (positions [])
                     : + fitnes < Alpha : core:
                             Alpha (core, Alpha pol = thress, politions Dell)
                     el: 1 fitures < Bela-score
                              Beta score; Beta pos = fitness, portions [:].copy
                      et: 1 Pitness & Delta score:
                             Delta score, Dalta por= fitness, positions (i) copy()
            Print (4" Iteration (iteration (i) / ( maxiter), But sore Hipal
            a = 2 - : foralion * (2/max_Here)
            for in range ( contratting outs)
                  for; in range (dim):
                        r, 12 = np. random. rand(), np. random rand()
                        A1, (1 = 2 " a + r-a, 2 + re
                         D-pulpha = abs ( c1 * Alpha porc;) - pasitions [iji]
x1 = Alpha posc; ] - Al * D-alpha
```

1, 112 = np roudom, roude), np roudon, roude) Az, (2 = 2" a + r-a, 2 + r2 x2 = Bola- pos[j] - A2 + D- beta T, T2 = np. random. rand(), np. random. rand() A3, () = 2+ a+ r- a, 2+ r2 D_della = abs (13 " Odla post;] - positions [isi]) 113 = polta-pos[] - A3+0-dola Positions (: 1) = (x1+x2+x3)/3 return Alpha-pol, Alpha-score det sphere-tendion (x): return up. sum () (** 2) dim = s scarch algents = 30 max: for = 50 16, ab = -10, 20 best position, best - score = gw o [sphore function, d'm, search gy more_iter, (b, ub) print ("Best position", best polition print ("Best score", but score) Bect. Botton 1-18212 -1,9555 1 4038 -1.7118 1.7226 output. Bert-Score : 1.4007

```
Code:
```

```
import numpy as np
def gwo(obj_function, dim, search_agents, max_iter, lb, ub):
  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")
  positions = np.random.uniform(lb, ub, (search_agents, dim))
  for iteration in range(max_iter):
    for i in range(search_agents):
       positions[i] = np.clip(positions[i], lb, ub)
       fitness = obj_function(positions[i])
       if fitness < Alpha_score:
         Alpha_score, Alpha_pos = fitness, positions[i].copy()
       elif fitness < Beta_score:
         Beta_score, Beta_pos = fitness, positions[i].copy()
       elif fitness < Delta score:
         Delta_score, Delta_pos = fitness, positions[i].copy()
    print(f"Iteration {iteration + 1}/{max_iter}, Best Score: {Alpha_score:.6f}")
    a = 2 - iteration * (2 / max_iter) # Linearly decreases from 2 to 0
    for i in range(search_agents):
       for i in range(dim):
         r1, r2 = np.random.rand(), np.random.rand()
         A1, C1 = 2 * a * r1 - a, 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, C2 = 2 * a * r1 - a, 2 * r2
         D_beta = abs(C2 * Beta_pos[j] - positions[i, j])
         X2 = Beta_pos[i] - A2 * D_beta
         r1, r2 = np.random.rand(), np.random.rand()
         A3, C3 = 2 * a * r1 - a, 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
  return Alpha_pos, Alpha_score
def sphere_function(x):
  return np.sum(x**2)
\dim = 5
search\_agents = 30
max iter = 50
1b, ub = -10, 10
best_position, best_score = gwo(sphere_function, dim, search_agents, max_iter, lb, ub)
print("Best Position:", best_position)
print("Best Score:", best_score)
```

PARALLEL CELLULAR ALGORITHM

```
Parallel Cellular alporthum
   import numby as up
   det optimization-function (position)
          return position Eo] + 2 + position [2] + 2
   det : miticalize - parameters ()
          grid-cize= (10,10)
          num: foration = 100
          heighbornhood - lise = 7
          retur-grid-size, numitications, neighbourhood tize
   det initialize :population (grid-fize):
         explation = uprand. uniform(-10, 10, logitid - 1:20 EO), grid-lize
          return population
 det valuate - fitness (population):
        filmen = np. zeroes ((population.shape [a), population.shape [i])
         for i in range (population. shape [0])
             for ; in range (population, chappe [1]):
                 fitness E; i) = optimization function (population []
         return Atness
det up date - states (population, tit nex, neighborrhood - size):
     updated-population = npcopy (population)
     for i in range (population, stapeEs):
        for ; in range (population. Shappeed):
              x-min= max (i - neighbourhood = rize, 0)
               x-more = min(: + neighbourhood_cize+1, population.chape
               d- 1, m. = max (? - uidypont bood - lise 10)
               y-nowx = min() + neighbourhood - size+1, population. shapel
               best-neighbour = population [: ]
               Cill Rints = titnes [1,2)
                for x in range (stmin, stmax):
                    for y'm range (y min, y marx)
                         it fitness Exchoder = total fitness:
```

```
explated-population (1,1) = (population (1,1) + best netyllown) (2
             return updated population
     det parallel cellular -aborithme):
              grid-size, numiterations, neighborhood-size= initialize promobers
               population = in/tialize +population (grid-(120)
                begt - solution = None
                 bett + it ness = stook ("in!")
                 : ( proitored. run) spror n; roitored; r. of
                       fitness = evaluate - fitness (population)
                        min-titness = np.min (titness)
                        if min-fitness & best-titness:
                               belt-titness = min-titness
                                best-soldion = population Inp. unovelsindex (nparg
                                             -min(Pitners), fitness. Phape D
                        population = uplate-states (population, fitness,
                                        neighbourhood 19820).
                        print (f" Iteration & iteration +1/1: Best files & best fil
                                                                       nessy")
               print (fulbert solution of best-solution y, Best fitness: { best-sitness")
                return best-soldion, best 8: theses
if -- rame -- = 1' -- main -- ";
              parallel = cellular algorithm()
tug to 0
 Best solution: [-8.9792, -8.3952],
    Best Fitness: -34.7489
```

```
Code:
import numpy as np
def optimization function(position):
  return position[0]*2 + position[1]*2
def initialize_parameters():
  grid\_size = (10, 10)
  num_iterations = 100
  neighborhood\_size = 1
  return grid_size, num_iterations, neighborhood_size
def initialize population(grid_size):
  population = np.random.uniform(-10, 10, (grid_size[0], grid_size[1], 2))
  return population
def evaluate_fitness(population):
  fitness = np.zeros((population.shape[0], population.shape[1]))
  for i in range(population.shape[0]):
    for j in range(population.shape[1]):
       fitness[i, j] = optimization function(population[i, j])
  return fitness
def update_states(population, fitness, neighborhood_size):
  updated_population = np.copy(population)
  for i in range(population.shape[0]):
    for j in range(population.shape[1]):
       x_min = max(i - neighborhood_size, 0)
       x_max = min(i + neighborhood_size + 1, population.shape[0])
       y_min = max(j - neighborhood_size, 0)
       y max = min(i + neighborhood size + 1, population.shape[1])
       best_neighbor = population[i, j]
       best_fitness = fitness[i, j]
       for x in range(x_min, x_max):
         for y in range(y_min, y_max):
            if fitness[x, y] < best_fitness:
               best_neighbor = population[x, y]
               best_fitness = fitness[x, y]
```

```
updated_population[i, j] = (population[i, j] + best_neighbor) / 2
  return updated population
def parallel_cellular_algorithm():
  grid_size, num_iterations, neighborhood_size = initialize_parameters()
  population = initialize_population(grid_size)
  best_solution = None
  best_fitness = float('inf')
  for iteration in range(num_iterations):
     fitness = evaluate fitness(population)
    min fitness = np.min(fitness)
    if min fitness < best fitness:
       best fitness = min fitness
       best solution = population[np.unravel index(np.argmin(fitness),
fitness.shape)]
    population = update_states(population, fitness, neighborhood_size)
     print(f"Iteration {iteration + 1}: Best Fitness = {best_fitness}")
  print(f"Best Solution: {best_solution}, Best Fitness: {best_fitness}")
  return best_solution, best_fitness
if _name_ == "_main_":
  parallel_cellular_algorithm()
```

GENE EXPRESSION ALGORITHM

```
gene expression
   import numpy as nos
   def optimization touction (soldion)!
         return solution [0] 2 + solution [1] 2
  det initial trze : parameter s():
        population - size= 50
        mumgenes = 2
        metalion rate = 0.1
        Crossonor - rate = 0.8
        num generations = 400
        return population-lize, nungares, nutation rate, crollowing
                  num-goverations
 det initialize-population [ population - size, num- genes).
       return reproviden. uniform (-10,10) population-size, nunequally
of ct cratuate - titness (population):
       return reparcy ( coptimization-function ( and ) for ind in popular
dot select-parants (population, litness):
       probabilities =1 / (fitness + 1e-6)
       probabilities = probabilities suport)
       ind; (1) = np. random choice Elan (population), size = len (papulation)
                   p = probabilities)
        roturn population (indices)
det crossover (parants, crossoverarale):
        CJ= Burd1840
        for i in range (o, len(parents), 2):
             it i, ti < len (parents) and np. random - rand(1 exollor
                  point = np. random. random+ (1, max. stope=1)
                   of & spring 4 = np.concade nade (( parenty (; , : port))
                                  (C: +20911+i)
                   offspring 2 = NP concate rate ( prout [:+1]:point).
                                   parents (1, point : )))
```

```
extend [Lottsping 1, offsping ]]
          else:
               ( Egring 1907 & gring the ) broke . gring the
                                  parents [1] (1+13 thrown) (132 Aurun)
                          (en (parents) else parants EJJ)
       return parray (off spring)
                              John wp away
  det mutate (offspring, matalian-rate):
          for individual in othering.
                 if uprandom. rand () < medation-rate;
                      gone = np. random. randint (individualsize)
                       inidividual (gene) = np. rand. ramallos i)
             return offspring
det gene-expression (population).
         retur population
det your expression atgorithme ?:
   population-1; 20, run -gones, mutation-rate, cross over-rate,
      nungenerations = nitalize sparanter
      population = initial '12e - population (population-size, nungane)
      best-solution = None
       best-titres = float ('int')
       for garaction in range ( name aprecions):
           1/h211 = evaluate - tituel (population)
            mistor - titres - fdex = nporg min(fitnell) a
            : & titness [min-fitness=idx] < best-fitness:
                    best titness = fitnessErinfitneskidso
                     best-soldion = population & min- 8: the 11-idx
             parents = select - goren's (population, githell)
             offipring = Erross over (parents, cross over-rate)
              population = modele (oterping, modelion rate)
```

population = gone - expression (population) print (f' evararation toponeration + 1 4 : BEH fitne 11 =16 print (f'Best rolution: {best-rolution}, Best Fitnes's best 416 return best-solution, best-fithers "++ -- name -- = "-- nain--"} gene expression - adgor, the Offety output: Best solution: [-7.9039 -9.968] Best fitness: -35.7441

```
Code:
import numpy as np
def optimization function(solution):
  return solution[0]*2 + solution[1]*2
def initialize_parameters():
  population size = 50
  num\_genes = 2
  mutation rate = 0.1
  crossover\_rate = 0.8
  num\_generations = 100
  return population_size, num_genes, mutation_rate, crossover_rate, num_generations
def initialize_population(population_size, num_genes):
  return np.random.uniform(-10, 10, (population size, num genes))
def evaluate_fitness(population):
  return np.array([optimization function(ind) for ind in population])
def select parents(population, fitness):
  probabilities = 1 / (fitness + 1e-6)
  probabilities /= probabilities.sum()
  indices = np.random.choice(len(population), size=len(population), p=probabilities)
  return population[indices]
def crossover(parents, crossover_rate):
  offspring = []
  for i in range(0, len(parents), 2):
     if i + 1 < len(parents) and np.random.rand() < crossover_rate:
       point = np.random.randint(1, parents.shape[1])
       offspring1 = np.concatenate((parents[i, :point], parents[i + 1, point:]))
       offspring2 = np.concatenate((parents[i + 1, :point], parents[i, point:]))
       offspring.extend([offspring1, offspring2])
     else:
       offspring.extend([parents[i], parents[i + 1] if i + 1 < len(parents) else
parents[i]])
  return np.array(offspring)
def mutate(offspring, mutation_rate):
```

```
for individual in offspring:
    if np.random.rand() < mutation_rate:</pre>
       gene = np.random.randint(individual.size)
       individual[gene] += np.random.normal(0, 1)
def gene_expression(population):
  return population
def gene_expression_algorithm():
  population_size, num_genes, mutation_rate, crossover_rate, num_generations =
initialize_parameters()
  population = initialize_population(population_size, num_genes)
  best solution = None
  best fitness = float('inf')
  for generation in range(num_generations):
    fitness = evaluate_fitness(population)
    min_fitness_idx = np.argmin(fitness)
    if fitness[min_fitness_idx] < best_fitness:
       best_fitness = fitness[min_fitness_idx]
       best_solution = population[min_fitness_idx]
    parents = select_parents(population, fitness)
     offspring = crossover(parents, crossover_rate
    population = mutate(offspring, mutation_rate)
    population = gene_expression(population)
    print(f"Generation { generation + 1}: Best Fitness = { best_fitness }")
  print(f"Best Solution: {best_solution}, Best Fitness: {best_fitness}")
  return best_solution, best_fitness
if _name_ == "_main_":
  gene expression algorithm()
```