VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.



LAB RECORD

Bio Inspired Systems (23CS5BSBIS)

Submitted by

SIDHVIN VIDYADHAR BURLI(1BM21CS211)

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 **Sidhvin Vidyadhar Burli(1BM21CS211)**, 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. Jyothi S Nayak
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	29-34
7	12/12/24	GENE EXPRESSION ALGORITHM	35-39

Github Link:

https://github.com/SidhvinBurli/BISLab

Program 1

Genetic Algorithm for Optimization Problems

Algorithm: P 3/10/24 Genetic Atgositerus Puport random POPULATION SIZE = 100 GENES = " abcdefghijrlmnopgrstuvwxyz ABCBERGH IREL WNOPBRSTUUNXYZ 12 3456789 0 , :! "# 8/(1 TARGET : "bmse &" class Individual (object): def _init_ (self, chromosome) self cheomosome = chlomosome Self. fitnon = Self. call-fitnen (1 (a dan method def mutated genes (s. 18): global GENES

gene = landom. choice (GENES) a clannethod def create-gnome (celk):
global TAPGET gnome len = len (TARGET) return [self.mutated_genes () for_in range (gnome_len)) definate (self, par2): child-thromosome = [7 for yp1, gp2 in zip (self. chromosome, par). prob= random. vaudomi)

The state of the s	
if prob < 0.45: child_(heomerone, append(gp1)	1
if moh < 0.9	8
if prob < 0.9 i child-(homosome. append (gp2)	
Manager and Anna Company of the Comp	
child-cheonarone append (sect	
milated give (1)	
etun Individual (child-cheomosomy)	
	A-Cort In-
def cel-fitners (self)?	0
global TARGET	A
filmers = 0	
for gs, gt an zip (Self. revolutione, TARLET:	(200
if gs!=gt · filment=1	
ochum fit non	1
def main ():	- China
global POPLIETTIONI-SIZE	
genectivs =1	Normal In
fourd = False	Trade
population = [7	Caralle S
des everla genera (coll)	
for in large (populationu-SIZE):	
gnowe = Individual. (seati-gnome () population append (Individual (gnome))	
population append (Individual (grown))	A CONTRACTOR ACC
estable as at Governord?	d
population = Sosted population, key=lands da	Ger
n: a fetners)	Gu
if population [o). Fitners (=0;	Ger
found = Frue:	Gu
briak.	Gen

	- man
2	Core Orange
(gp2)	new gen = [] s= int ((10* population.size)/100) new gen. extend (population [:3]) s= int ((90* population.size)/100))
set.	for 3n range (s): parl = random. (hoice (population 1:50)) par2= landom. (hoice (population 1:50)) child= parl. morte (par2) new: gen. append (child)
4	population = new-gen
TARLET	point ("Gen: 1) It String, 53 17 fitners of 3" format (generation, "" you (population fo) clusomosome), population (o) fitners)?
4	generation += 1
	print (generation of 1 string 12) the
	format (generation, " ", jan (population / obesses) population (0) fither))
w)	maine,
ulo da	Gen: 1: Str. BRGCp fitnes: 3 Gen: 3 Str. EQSIEE Fitnes: 2
	Gen: 4 Str: BHSCE fitners: 1 Gen: 5 Str: BruscE fitners: 0
	AND THE RESERVE OF THE PARTY OF

```
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 \leq 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:

```
alioha
   Particle Swagn Optimization
         import random
        Priport math
         Emport copy
        import sys
          det fitner rashigin ( position).
              - return sum | xixx = 10 x math. los (2 math. pi xi)+
                             to for xi'in position)
         det fithen sphiel (position):
              return sum (xixx2 for xiin pouton)
         class Particle:
              def inst _ (self, fitner, din, minx, maxx,
                  self. md = random. Kandom (seed)
                  Belf position = | self . and uniform mink, marx)
        for in lange (dim)]
                  self velocity = | self and uniform (mink, mars)
                     for in longs (din)?
                   self fitnes = fitness (celf. position)
                   SHIP. HE
                   self. best_part-pos = copy. copy (self. position)
Self. best - part-fitness Val = Self. fitness
        det pro (filmer, maxiter, n, din, minx, maxx):
              W, (1, (2 = 0.729, 1.49445, 1.49445
               Swam = [particle (filmen, dim, minx, maxx, i) for

P in longe (n) ]

best-swam pus = copy. copy (example) position)
```

best-swarm-fitnes Yal = min (p. fitner for p in swann) for p in swam; If p.fitness < best swam folianval: best swarm. Formen Val = p. fitners best swamp pos = cupy (p. position) For Iter in range (max-iten): 9f iter 1.10 == 0 and iter 71: print (f" ster= (14) bd fitners = of best swan fitner Val 1.34% for p in swarm: 31,72 = sandom laudon (), randomlaudani) for k in longe (dim): p. valocity 187 - (wx p. xulocity (k7 + er x) x/p. bent-part-posik) p. porition (K7) + (2. × x 2 * (best - avoram. puslk) - p.podk) p. velocity [t] = max(min(p. velocity)ki maxx) ninx) for kin long (bin) p-position[r] + = p velocity (x) p Fitners = Fitners (p-position) if & fitners & D but part - Fitner Kill p but part fitner Nat = p. ftner p. best part-pus = copy lopy (p. postion)

P. fitners < hert swarm-fitnersel' blot svans potner Val = p. fitnes nom. pos = cop g. copy (p. positive)

returns bet-sweetin-pos point (Bungis PSD) fitner = fitnes - rastoign num particles, max-iter = 10,50 Dest-position = psoffetner, max-iter, num-parbala, din, whited full sole 12 1 -50, 50) (" In Bort Soluntron found, ["1.6 f' 1. pos for pos in bent partion ?) "Fitness of but soln = 1.6 f" -1. fitner (beet position)) Begin pso Hu = 10 best Pitner = 25 934 12,294 1th = 19 best fitners = 12.294 best-fitners = 6.197 2tu=40 best forms = 2.199 Iter=50 bet fitues = 1.830 Best Solution found: [10.013356 '0.058029' -0.96840) form of Pest solo: 1810,272 hat some one constant

```
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()
         p.velocity[k] = w * p.velocity[k] + c1 * r1 * (p.best part pos[k] - p.position[k]) + c2 * r2 *
(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 fitness Val:
```

```
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)
```

PROGRAM 3 ANT COLONY OPTIMIZATION

Algorithm.

Algorithm:	
1	Pour 14/14/24
- OBCOS	And colony Optimization
1	Purport random
	emport numpy as np
1	mpost math
- Allan	The section of the
	To the state of th
· ·	class Eity.
6	def_init_ (self,x,y).
AV	Belf. x = x
	self. y = y
	det distance (self, city): return math sgot (1 self. a-aty. x)**2+
2	return math sgot [1 self. a-aty.x) 2+
	(exit, y-city, y) **2)
) Market	
- A FAREIT	clem ACO-TSP
Carry Am	def _ init _ (self, cities, num-onts, num-iterations,
	alpha=1.0, beta=2.0, eho=05, 90=0
	self. cities = cities
	self. num_iterations= num_iterations=
The said	self. alpha = alpha self. beta = beta
A	self. sho = sho
	self-90 = 90
	self. num. cities = len (101ties).
	self. phenomone= np. ones (1916 num with, celk now
7	-cuties))
The Continue	self. heuristic = np. zeroes (1 self. num cities,
MAG	GIV. num-cities))
To For June	and hast true = "None."
	Self. best-toue = float ('inf').

for i in range (self num - citres) s for j in lange (Ift, 5°18, num cities) dint = cities[in distance futres(7) self shunistration[j] = 1.0/dist sy dist! =0 else 0 Silf hemeritare [i][] = Self hencertifile det select next city (self, werend city, visted city);
probabilities = np zeros (self brem-lities) total phelomone = 0.0 for city in large (self. mun-cities);
if city not in visited-cities; phenomore = self phenomere (went city) heuristic = seef heuristic (auch city) Coty ? peobabilities (ity) = pheomone *heueistiz tota phicomere + = pedalotter [city] if total pheconon == 01 ectuen sandom thone (Faity for city in lange (self numcities) of city not is probabilities /= total pheromous if Lendon, Lendon () < Self-90; next-city = np. argmar (podbabilities) next-city: mp. landons. chair sett. non whis, O- publishitiii)

```
ethern next-city
     def update-pheromone (self, ants):
          Self-phecomone » (1- selfeho)
            for ants in ants:
                 pheromone - deposit = 1.0/ant. tour-length
       for ir in sange (self num_cities):
             current - city = ant. tour(i)
                      next-city = aut. tous [(i+1) f. s. 1. nun-
                                                citics 7
                     SIIF. phosomore [curent city] [next-city] +=
self. phosomone [nent-city] (coect-city) +=
                                      physomore-deport
    clan Ant
           der _init - (self, neur-citres, 100-top):
                  suf num-cities - num-cities.
               suffaco typ= aco-typ
                   Self. tour = []
                   cer. tour-longth = 0.0
                construct - solution (sett).
                 stall-city = landon, landint (0, self. numertia-1)
                  self. tour = [start city]
                   visited_cities = cet (self tours)
                  creent_city = start city
                   while her [selfstow) < self neur cities
                   nept-city: sect aco top. septet next city
                                          auchedy, vistherast
```

self tall append (near tity) self. tale length + = self. aco_tsp. citing (web-city) elistance (self aco-top cities [next idy]) Self-toue-deligth + = 80 H. aco-tsp. cities [self-tour [-1].

Ly Hancel self. aco-tsp. cities [self-tour taux[0]] id name = " mais " : cities = [City (0,0); City (1,3), City (4,3), (3,16,17), alo = 10,75p (citis = citis , numants = 10, 50,10, 20,00 post tour , but to me length - aw. eurs) point ("But pour , but tour) print (" Best pone lougth", but touchength) Best tous (9,0,1,2,3) But tou length: 15.1529

```
Code:
import random
import numpy as np
import math
class City:
  def init (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:
            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)
```

PROGRAM 4 CUCKOO SEARCH ALGORITHM

Algorithm:	
	Cuckoo Search 21/11/24
	110
ut-city)	Simport number as me
idu	from scipy special sind
idy)	from scipy special report gamma
	det objective (x):
	det objective (x): setum np. sum (x**2)
Durch Janet	The state of the s
taus(o))	art levy-flight (beta, dim):
	sigma = (gamma (1+b.ta) * np.sin(np.pi* beta/2) / (gamma ((1+b.ta)/2) * beta*np.power
ZV	(gamma ((1+bota)/2) beta np.power
	(2) (1-st -1)/2) 11 × ((1)-st)
60	V = np. random. normal (0, sigma, din)
16 13	- np. random. normal (0, 1, din)
16,17	rolum u/ np.abs (v)*x(1/beta)
1 Jarot	det control seach (als G As bound Also a control
,20,05	det cuckoo seasch (obj.fure, dim, bounds, N=20, pa=0.25, max_iter=100):
AR	
AX	Titnes = no assau (c abi form (next format)
2	fitness = np. oriay (pobj famil nest for nest a)
A	best-nest = nests [np. aigmin (fitnen)]
	best-fitness = op. min(fitness)
	for in range (max-itr):
A A	for in range (max-itr): new nexts = np. copy (nexts)
1 Car.	tor I in Grange (N):
	step = livy-flight (1,5, dim)
	-nest & n.
	new_nestr (i) = nectreli7+ 0.6) * step
	new-nests [i] - no. clip (new-north) bounds al,
	[bounds[1])
	new-fitners = np.array (16 by-func (next) for
	nest in new nexts])
BOOK STATE	

100 Buc for i in lange (N):

If nprandom rande) < pa and

new fahers (i) Cfether (i):

nexts[i]= new nexts[i) fitnen [17 = new fitners[i] best-nest id x = sp. arymin (fitner)

best-nest = nests

but formers = fitners [best-nest 1d x] inchem best pest fetners priof (f" best feet forment (") point (f" Bost fitners: (best fotners 4) Best nest: [2.519, 0.8] 2.396 1.168 -1.560) but fitners : 32 12 40 2 to 0 of filebox = (?) stem colf

```
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}")
```

PROGRAM 5 GREY WOLF OPTIMIZATION

Algorith

Algorith	ım:
	I O bee
3	
	Grey LIGHT Optimization 2 stutza
A CONTRACTOR	Emport numpy as np
A	3200 0 7 2
Dered	July July server carety
1	
	Bete-pos = np. zuos (dim)
	Detta-pos = np. 2100s (dim)
3/16	-Alpha-sc = float ('inf')
	Bete-sc = Hoat ("inf")
Y	Petta-sc = float ("inf")
Till-co-	The second of th
Ų.	pos = np. Landom- witom (16, ub, (scarch-azerts, din),
VIII.	(((((((((((((((((((
Y	for it êtrin range (max-iter)!
	for i in large (seasely agent);
Mbusto	pos [8] - np. clip (possin 16, ub)
2 2 53	pos[P]- np.clip (possi), lb, ub)
A Casta	fitness = Obj fan (possi)
	AHOLOVER . II SALAHAR AY
2000	if fitness < Alpha-sc 1
	if fitness < Alpha-sc; Appra-sc, Alpha-pos= fitness,
16 350	pus (i). cop y ()
	advan Alphanon Alphanol
	elif -fituro (Beta_s come:
	Beta sc, Beta pos = filmesi, posli Tropy()
	E (Present & more and in
	elif filmen & Duta.score
	Delta Sc, Del ta pos - fitners, possi) copyl)
	Fig. (2) (1) (1) (2) (1) (1) (2) (1) (1) (2) (1) (1) (1) (1) (1) (1) (1) (1) (1) (1
	point [f" Iterator lits + 17 (max-iter), But Scare
	ont (+ " Alpha-Sinci 6 / ")

20000 a= 2 - itr x (2/maxites) for i in sange (search_aques): for 1 is sange (dim):

82,72 = np. sandom, sand (), np. landom and A, c1 = 2 x x 1 - 9, 2 x 2 D-alpha = abs (Clädlpha pos (j)pos (i,j))

XI = Alpha - pos(j)-41 P-alpha 89,72 = np. Landon, rand (), np. sandon send ()
A2, C2 = 2 x a x 81 - a, 2 x 2 D. beta = abr ((2* Beta. pod j) - posli, j])

42 = Beta-por [j] - A2 D. Deta 13, (3 = 2 ng. Landomsand 1), ng. landomsand11 13, (3 = 2 na no 1 - a, 2 no D'delta = alse (c3 alta-pos(j) - pos(i,j))

×3 = Pelta-pos(j) - A3 D-delta pos(1,j)=(x1+x2+x3)/3 rotum Alphapos, Alpha sc def spher function (x) return np. sum (x 2): Stallhagents = 50 mer-Ita = 50

	C
	Part of College A Parties
- W	Dest-pos 1 De 1751 = gwo 5 pure further, duin soull and
1	best-pur ibests: = gwo(spun funter, duin, sour ogd,
andon mul	Devit ("Best Docition", "
	print ("But - Sicre; "; best position)
	Dest-siere; Dest. ser)
([[,	Continues and star late
3,1)	Output (moi) = welle
	now = gettingen
	Best Slove: 4565059 4.7615e-09
lend()	Best Pos: [-2.9829 e=05, -2.911 e-05 +2.9204e-05-
	317020-05
(,j))	al stulus
(1000)	3 1702e-05]
	112: 112:12 7: 15
condl	malalage marks
(1,57)	Contiduo lana 1 to lan 355
(1,1)	· Pohyale mil Supre] most que = 30 53
N TOT	Degester of stage
	(e) ag orle mattalages agores (e) is (e)
The state of the s	INVENDED AND AND IN THE
	- Balling & Consordi
1	Se bron Francis
The same	summer, result, and along of dance of the get
The state of the s	Contra de la contra del la contra de la contra del
	Contract policies of ATMO
-	the one continue I speece his i say
	The state of the s

```
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 j 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[i] - 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[i] - positions[i, i])
          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[i] - 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)
```

PROGRAM 6 PARALLEL CELLULAR ALGORITHM

Algorithm:

Algorithm		
P 100		
	Parallel Collular Algorithm	
Linkson!	This sale of a large force of the decog find	- Andre
	Emport numpy as no	V F
4.3.5	clef opt-func (pos)	
	clef opt-func (pos) actum posto7**2+ post 11**2	
	def initialia: parameters ():	for many
	grid-size = (10/10)	
	nun-iter = 100	
	neighbourhood siz = 1	
- 2 - 42000 C+	octum guid-size; numiter, hushberhood-size	
	det initial_pop (gold_size)	
	population = np. ravidous. un for (-10, 19 (gridse, [0], grids/ze[1], 2))	N
	[0], grid stze[1],2))	
	setum population	
	det evaluate fitners (population)	N
	fitness = np. zeros ((population. shapelos),	
	population shapeli)	
	for i in range / population. shape(6))	
	for ; in range (populationshape[1]):	n
	fitness(i',j) = opt-tunc/population[i]	4
	se him fitners	No.
	det update-scored (population, fitness, neighbanhad 52	
	update population = np. lopy (population)	
	tor I'm lauge (population shape [0]):	
TO THE	for j in range (population. chape[1])	No. of the last
1000	·	No.
	7 -min = max (i - reighbaulood-sec, 0)	

y max = min (it neighbourhood- sie + 1, Population shape(0) y-min = max(j - neighbourhoodsize ,0) y max = min () { neighbourhood size + 1, populationshipsi) best neighbau = population (i, j) best fitner = (itners (i,j) for x in lange (n.min, x max): for y in large (ymm, y max): best-fitner = fitners [21,4] aparted-pop [i,i) = (population[i,i]+
best-nuzhioue)/2 octurn updated pop det pagettel_coll_algo() gold-size, near iteretion noighbauhard-size initialize_ parameter (population = "Initialize - pop (goid-size)
but sol = None
but fit = floot (inst) for iteration in sange (num-iterature): filmen = evaluate-frimen (population) mM-fitnen= np. mm (fitnen) If MIN-Fiture Derrit?

bost: fit = population [np. unsand indu (np. augmin (form), fother Shape)] population = update states (population, Films, point (3' Heration literally: best fitner: Abest fit; If name -= "_mah_"; preallel cellular algorithm () Best Solution: [1.0464e-25 1.243e-25]
Best form: 2.7242e-50

```
Code:
import numpy as np
def optimization function(position):
  return position[0]*2 + position[1]*2
definitialize parameters():
  grid size = (10, 10)
  num iterations = 100
  neighborhood size = 1
  return grid size, num iterations, neighborhood size
definitialize 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(i - neighborhood size, 0)
       y max = min(i + neighborhood size + 1, population.shape[1])
       best neighbor = population[i, j]
       best fitness = fitness[i, i]
       for x in range(x min, x max):
         for y in range(y min, y max):
            if fitness[x, y] \leq 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()
```

PROGRAM 7

GENE EXPRESSION ALGORITHM

Algorithm:

gorith	m:
	O best O
	Page C
	Optimization vid Gene Expression Algorithms
1	De la
3	Amport numpy as up
11	A STATE OF THE STA
	def opt force (solution): return solution[0] 2 + solution[1] 22
100	Solution of Solution (1)
100	der initial-paral);
-	Pop size =50
-100	Niemgenes = 2
- 60	met rate = 0.1
	(ros : rate = 0.8
- 4	hum gen = 1000
-	return pop size, num-genes, must-rate, mosse
	nun-gen
	def intel pap / pap (ize num-gines).
	def intal-pop (pop size, num-gener): ve turn np raudom unitoré: 10,10, (pop size, nu
	def evaluate-fit (pop):
	· return np may (popt func (ind) for in
	in pop])
	(what is a transportant or an isother
	def solect-parents (pop, fitners)
	prob = 1/(fitness + le-6)
	prob/= probsum()
	indizer = np. landom choice (len(por
7 7 0 2 3	9170 = 1200 11
	ochem popsindices
- 311	The state of the s
	7 3 10 12
-156	

det (nossoree (parcets, crossovierate): offspring = [7 for 1 in range (0, len (parents), 2) if it (but (parus) and mp. laudon xund) consider rate: point = pp. sandon sandint (present snapel offspring = np. concatenate (1 parents (in point)) offsporme 2 = mp. concetenate ((pants [i+1; point) parents [i, point i)) offsporks. extend ([offsporks 2]) else ; Offspray. extend [paints ?] paint [i+1) 1 & len (preus) setaron repossary (observers) gene expression population) gate, expressory algo (): pop sile, neun genes, mustatres sets, journants, neun = instillar parameters population = limited size - pop. (pop-size 14m gury

Prop () beth sol - None werd fit = floot ("int") for gen in range (num-gen): fitnes = lyduate filmen (population)

fit-idx = mp.arg. mu (fatur)

if follows ffd-idx) < best getness:

best follows - filmen [fit-idx]

best sol = pop (fit-idx)

parents = select parents (pop, fitness)

offspring = crossore (parents, crossonsele, pop = generation (pop) point (I' Bert : < bert-sol & Bert Februs setum bet sol, best fit geneexpris algo () Bert - Fit: -35.744,

```
Code:
import numpy as np
def optimization function(solution):
  return solution[0]*2 + solution[1]*2
definitialize 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
definitialize 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])
       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:
       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()
```