5E

'BIS LAB'

INDEX

Name Setu Mishra Std V Sec E

Roll No. Subject BIS LAB School/College

School/College Tel. No. BMSCE Parents Tel. No.

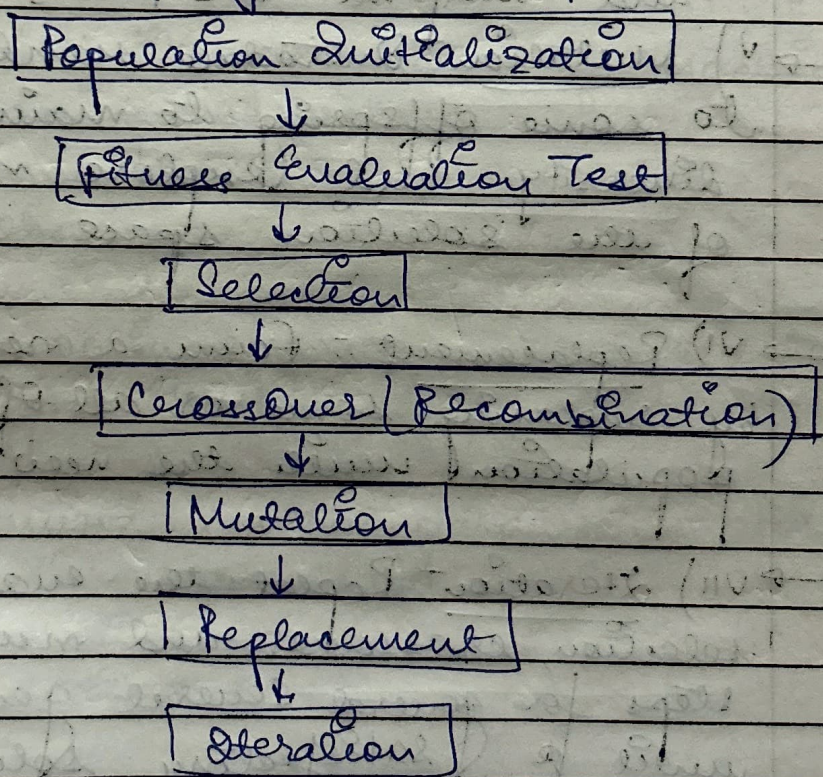
Sl. No.	Date	Title	Page No.	Teacher Sign / Remarks
		~~Implement the following game.~~		
	28/9/24	Summary of all algorithms		
1.		Genetic Algorithm for optimization problems.		10
2.	3/10/24	Particle Swarm Optimization for function optimization.		10
3.	24/10/24	LPseudocode (PSO)		
4.	7/11/24	Ant Colony optimization		9½ =10
5	21/11/24	Cuckoo Search (CS)		10
6	28/11/24	Grey wolf optimizer (GWO)		10
7		Parallel Cellular Algorithms and programs		10
8		Optimization via gene expression algorithms.		9

Date __/__/__
Page _____

Lab-1 : Genetic Algorithm for Optimization Problems

(*) Genetic Algorithm is a method for solving both constrained and unconstrained optimization problems that are based on natural selection, that process that drives biological evolution. The genetic algorithm repeatedly modifies a population of individual solutions.

The main algorithmic steps are:



→ i) Population Initialization - Start with a randomly generated population of potential solutions, often represented as strings (like binary or real nos.)

→ ii) Fitness Evaluation - Evaluate how well each solution performs, relative

Date __/__/__
Page _____

to the problem's objective.

→ iii) Selection - Select the best-performing solutions to be parents for the next generation. Eg - Tournament selection, Roulette wheel selection

→ iv) Crossover - Combine pairs of parents to produce offspring. This mimics biological reproduction where parts of each parent's genetic material are mixed. Eg:- Single point, two point and uniform crossovers

→ v) Mutation - Introduce random changes to some offspring to maintain genetic diversity and explore new areas of the solution space.

→ vi) Replacement - Form a new generation by replacing some or all of the old population with the new offspring.

→ vii) Iteration - Repeat the evaluation, selection, crossover, and mutation steps for several generations until a satisfactory solution is found or a predetermined condition is met.

Applications:

→ (i) Optimization problems -

- Used for optimizing structures,

Date / /
Page

components and systems in field like aerospace and civil engineering.

- Helps in optimizing the allocation of resources in industries like telecommunication and logistics.

(2) Machine learning -

- Assists in selecting the most relevant features for improving model performance
- Optimizes the parameters of machine learning algorithms.

(3) Robotics

- finds optimal paths for robots to navigate environments
- Optimizes control parameters for robotic movements.

x

03/10/24

Date __/__/__
Page

Lab 2:- Particle Swarm Optimization for Function Optimization

- Particle Swarm Optimization (PSO) is a popular optimization algorithm inspired by the social behaviour of birds and fish. Its particularly effective for function optimization, especially in complex, multidimensional spaces. Its mainly for function optimization.

- Algorithm:

- Initialization - Set parameters such as the number of particles, maximum iterations, cognitive and social coefficients, and the search space boundaries.
- Initialise the positions and velocities of all particles randomly.

- Fitness Calculation - Evaluate the fitness of each particle using the objective function.

- Updating best positions - Each particle keeps track of:
- personal best position (pBest) - the best position it has encountered so far.
- global best position (gBest) - the

Date / /
Page

best position found by any particle in the swarm.

- Velocity update - Update the velocity of each particle based on its own best-known position and the best known position of the swarm:

$$v_i = w \cdot v_i + c_1 \cdot r_1 \cdot (pBest_i - x_i) + c_2 \cdot r_2 \cdot (gBest - x_i)$$

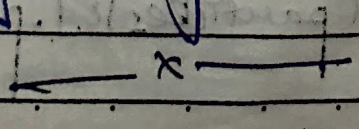
- Position update - Update the position of each particle based on its new velocity

$$x_i = x_i + v_i$$

- Boundary conditions - If a particle moves out of the defined search space boundaries, it can be reset to the boundary, or reflected back into space

- Termination criteria - The process repeats until a stopping criterion is met like maximum no. of iterations, etc.

- Applications
  - Mathematical function optimization
  - Machine learning
  - Signal Processing
  - Engineering Design Problems



24/10/24

Date __/__/__
Page _____

PSO - pseudocode

```
#Initialise parameters
num_particles = N
max_iterations = MAX_ITER
inertia_weight = w
c1 = cognitive_parameter   # personal best influence
c2 = social_parameter      # global best influence

#Initialising the swarm (particles)
for i in range(num_particles):
    particles[i].position = random_position_in_search_space()
    particles[i].velocity = random_velocity()
    particles[i].pBest = particles[i].position
        # personal best initialised to the starting position
    particles[i].pBest_value =
        fitness_function(particles[i].position)

# Initialise global best (gBest)
gBest = particles[0].pBest
gBest_value = particles[0].pBest_value

# Main optimization loop
for Iteration in range(max_iterations):
    for i in range(num_particles):
        # Evaluating fitness test of the current position
        fitness_value = fitness_function
            (particles[i].position)
```

Date __/__/__
Page _____

```
#update personal best (pBest)
if fitness_value < particles[i].pBest_value:
    particles[i].pBest = particles[i].position
    particles[i].pBest_value = fitness_value

#update global best (gBest)
if fitness_value < gBest_value:
    gBest = particles[i].position
    gBest_value = fitness_value

#update velocity and position of each particle
for i in range(num_particles):
    for d in range(num_dimensions):
        r1 = random(0,1)
        r2 = random(0,1)
    #update velocity
    particles[i].velocity[d] = (inertia_weight * particles[i].velocity[d]
    + c1*r1*(particles[i].pBest[d] - particles[i].position[d]) +
    c2*r2*(gBest[d] - particles[i].position[d])

#update position
particles[i].position[d] = particles[i].position[d] + particles[i].velocity[d]
```

Date __/__/__
Page _____

```
# Ensure position is within bounds
if particles[i].position[d] < lower_bound[d]:
    particles[i].position[d] = lower_bound[d]
if particles[i].position[d] > upper_bound[d]:
    particles[i].position[d] = upper_bound[d]
# Return the best soln found
return gBest, gBest_value
```

7/11/24

Date __/__/__
Page _____

Ant Colony Optimization for TSP

Ant Colony Optimization is a metaheuristic algorithm inspired by the natural behaviours of ants searching for food. In ACO, artificial "ants" simulate the behaviours of real ants that deposit pheromones on the ground to communicate and mark their path, influencing the behaviour of other ants. This approach has been successfully applied to various combinatorial optimization problems, including TSP.

Algorithm

① Initialization

- Initialise the pheromone levels τ_{ij} for all edges between cities (usually a small constant value.
- Set the parameters α, β, no. of ants, no. of iterations, pheromone evaporation rate, etc.

② Ant Tour Construction

- for each ant, start from a random city.
- At each step, the ant chooses the next ~~day~~ city based on the pheromone levels and distance.
- Repeat until all cities are visited and tour is completed.

Date __/__/__
Page _____

(3) Pheromone update
- After all ants have completed their tours, update the pheromones on the edges according to the rule
$\rho_{ij} = (1-\rho)\rho_{ij} + \Delta\rho_{ij}$ where
- ρ is the pheromone evaporation rate
- $\Delta\rho_{ij}$ is the amount of pheromone deposited by ants on the edge (i,j) which is inversely proportional to the length of the tour of the ant.

(4) Termination
- If the stopping criteria (such as maximum no. of iterations or convergence to the optimal solution) are met, stop the algorithm. Otherwise, repeat from step 2.

(5) Initialise pheromone matrix.
r[i][j] = r0 for all pairs of cities (i, j)
Initialise best tour = empty
Initialise best_length = infinity
for iteration = 1 to max_iterations:
  Initialise all ants' tours
  for each ant k:
    ant_tour[]
    current_city = random.city()
    ant_tour.append(current_city)

Date __/__/__
Page _____

while tour is not complete (all cities visited):
    next_city = choose_next_city(current_city, ant_tour)
    ant_tour.append(next_city)
    current_city = next_city
tour_length = calculate_tour_length(ant_tour)
if tour_length < best_length:
    best_tour = ant_tour
    best_length = tour_length
Apply pheromone vaporization (v[i][j] = (1-ρ)*v[i][j])
for each ant k:
    for each edge (i,j) in ant_tour:
        update_pheromone(i, j, ant_tour, tour_length)
Return best_tour, best_length

Function to choose the next city using pheromone and distance

function choose_next_city(current_city, visited_cities):
    probabilities = []
    total_pheromone = 0
    for each city j not in visited cities:
        pheromone = v[current_city][j] *

Date __/__/__
Page _____

```
visibility = (1 / distance[current_city][j]) ** β
total_pheromone += pheromone * visibility
for each city j not in visited_cities:
    pheromone = τ[current_city][j] ** α
    visibility = (1 / distance[current_city][j]) ** β
    probability = (pheromone * visibility) / total_pheromone
    probabilities.append(probability)
next_city = select_city_based_on_probability(probabilities)
return next_city
```

```
// func to update pheromone on an edge (i, j)
func update_pheromone(i, j, ant_tour, tour_length):
    Δτ = Q / tour_length   // Q -> constant
    τ[i][j] += Δτ
```

Applications:

21/11/24

1. Reservoir optimization
2. Water distribution systems
3. Urban drainage system design
4. Finding best possible route
5. Job resource allocation

21/11/24

Date __/__/__
Page _____

Cuckoo Search

Summary:

The Cuckoo Search Algorithm is a nature inspired optimization algorithm based on the parasitic behaviour of cuckoo birds. This algorithm is designed to solve complex optimisation problems by mimicking the breeding behaviour of cuckoos and their interaction with host birds.

Algorithm:

1. Initialise population of n nests (solutions) randomly
2. Evaluate the fitness of all nests
3. while stopping criterion is not met:
   a. For each nest:
      i. Generate a new solution using Levy flight
      ii. Evaluate the fitness of new solution
      iii. If the new solution is better, replace the old nest with the new one.
   b. Abandon some nests with probability p-a and replace them with new random solutions
4. Return the best solution found.

Date / /
Page

Pseudocode:

(1) Initialise parameters:

- N: No. of nests (population size)
- MaxIter: Maximum number of iterations
- Pa: Probability of discovering alien eggs
- Bounds: specific bounds for the search space (lower and upper limits)

(2) Initialise nests: (solutions)

- For each nest i = 1 to N:
  - ~~Evaluate fitness(x_i) using the objective function f(x_i)~~
  - Randomly generate initial solution x_i within search space (lower bounds and upper bounds)

(3) Evaluate fitness of each nest:

- For each nest i = 1 to N:
  - evaluate fitness(x_i) using the objective function f(x_i)

(4) Set best solution as the initial best solution:

- Set BestSolution = argmin(fitness(x_i)) for i = 1 to N.

(5) For each iteration (t = 1 to MaxIter):

(a) Generate new solutions (nests) using Lévy flights:

Date / /
Page

$x\_i' = x\_i + \alpha \oplus \text{LevyFlight}()$

- Ensure x_i' stays within the search space bounds.
  If x_i' is out of bounds, adjust it to stay within the boundaries
- Evaluate Fitness (x_i')

(b) Evaluate and compare new solutions with the current nests.
- If Fitness(x_i') < Fitness(x_i):
  - Replace the old solution x_i with the new solution x_i'.

(c) Randomly replace nests with a probability Pa (discovery of alien eggs)
- for each nest $i = 1$ to N:
  - with probability Pa:
    - Replace nest x_i with a randomly generated solution ~~x_i~~ x_rand within the search space.
    - Evaluate fitness(x_rand)

(d) Update the best solution:
- If a better solution is found:
  Update BestSolution to the best solution among all nests

(e) Return the best solution found after MaxIter iterations:
- Return BestSolution.

21/11/24

25/4/24

Date __/__/__
Page _____

Grey wolf optimization

Summary: is a nature-inspired optimization algorithm that simulates the social hierarchy and hunting behaviour of grey wolves in the wild.

Algorithm / Pseudocode:

① Initialise parameters

- N: Number of wolves (population size)
- MaxIter: Maximum no. of iterations
- D: Dimensionality of the search space
- Search space bounds: lower and upper limits.

② Initialise population of wolves

- For each wolf i = 1 to N:
  - Randomly initialise position x_i in the search space (within bounds)
  - Evaluate fitness of x_i: fitness(x_i)

③ Identify the top three wolves:

- Set x_1 = the wolf with the best fitness (alpha wolf)

Date __/__/__
Page _____

→ Set x_β = the second best wolf (beta wolf)

→ Set x_δ = the third best wolf (delta wolf)

(4) For $t = 1$ to Max_Iter:

- For each wolf $i = 1$ to N:
  - → Compute the distance to alpha, beta, and delta wolves:

$$D_\alpha = |C_1 * x_\alpha - x_i|$$

$$D_\beta = |C_2 * x_\beta - x_i|$$

$$D_\delta = |C_3 * x_\delta - x_i|$$

  - → Update the position of wolf-i:

$$x_i(t+1) = (x_\alpha + x_\beta + x_\delta)/3$$

  - Where C_1, C_2, C_3 are random coefficients used to balance exploration and exploitation:
    C_1, C_2, C_3 are random vectors in the range $[0, 2]$

- Update the fitness of each wolf and identify the new alpha, beta and delta wolves based on fitness

(5) If stopping criteria met (eg: Max_Iter or convergence), stop the loop.

(6) Return the best solution found: x_α (position of the alpha wolf)

28/11/24

28/11/24

Date __/__/__
Page

Summary

PSO Algorithm

PSO is a methodology inspired by the choreography of a bird flock. It is a distributed-behaviour algorithm that performs multidimensional searches. PSO belongs to the class of metaheuristic methods as it makes few or new assumptions about the solutions. The algorithm works by moving particles (structures) in a search space based on efficient algorithms that use the position and velocity of the particles. Hence, all the individuals in the swarm can quickly converge to the global position and a near-optimal geographical position following the behaviour of flock and their flying histories.

ACO Algorithm

The ACO is another swarm optimization technique based on the behaviour of ants seeking a path between their colony and a source of food. The original idea has since diversified to solve a wider class of numerical problems. Usually, ACO is applied for problems that

Date __/__/__
Page _____

exhibit frequent dynamic changes. The main advantage of ACO is the ability of the algorithm to run continously while adapting to changes in real-time.

CS Algorithm

The CSA algorithm is based on the cuckoo bird behaviour, which lays eggs in another host bird. The main goal of the cuckoo bird is to mimic the host bird's eggs and avoid being discovered by the host bird. Hence, the optimization goal is reached if the host bird cannot discover the cuckoo eggs.

Grey Wolf Optimization

GWO is a metaheuristic algorithm that is inspired by the behaviour of grey wolves in leadership and hunting. The algorithm classifies a population of possible solutions into four types of wolves α, β, δ and ω. The four types are based on the fitness of solution, which means that α is considered the best and ω is the worst. The new generation is created by updating the wolves in each one of these four groups. This update is

Date __/__/__
Page ____

based on the first three best solutions obtained from α, β, δ in the previous generation.

Parallel Cellular Algorithms

Parallel algorithms refer to a computational approach where multiple solution points or populations of solutions are worked at each iteration of the algorithm. By employing parallel processors, they can significantly enhance the speed of convergence of the final solution. They divide a problem space into smaller, localized units, referred to as cells. Each cell operates independently yet interacts with its neighbours based on predefined rules.

Optimization in gene expression algorithm

It often involves improving the computational efficiency and accuracy of methods used to model, predict or analyze gene expression. These algorithms are central to bioinformatics for tasks like identifying gene regulatory networks, finding biomarkers or understanding the dynamics of gene activity in different conditions.

29/11/24

Date __/__/__
Page _____

Parallel cellular Algorithms

Parallel cell algorithms are a class of algorithms where computation is distributed over a grid of cells, and each cell performs operations in parallel based on simple local rules.

Algorithm:

Step 1: Define objective function $f(x) = \sum x_i^2$

Step 2: Initialize parameters

Grid size: nxn grid where each cell represents solution

Dimensionality: No. of variables in solution (eg 2D)

Bounds: Defines range of sol^n space

Mutation rate: Probability of mutating a solution

Step 3: Initialize grid

Randomly generate initial sol^{n}s of each cell within the defined bounds.

Step 4: Evaluate initial fitness

Calculate fitness for each cell using obj function

Step 5: Main evolutionary loop:

For each iteration

Date __/__/__
Page _____

i) Select parents : choose a cell & best sol^n from neighbourhood (up, down, left, right)

ii) Apply crossover: Generate offspring

iii) Apply mutation: Randomly modify offspring with small probability to indicate diversity

iv) Replace sol^n : If offspring has better fitness than current sol^n, replace in grid.

Update grid

Step 6: Track best sol^n after each iteration

Step 7: Stop after a fixed no of iterations or if convergence is reached & return best solution and its fitness value.

Applications:

- Image processing and computer vision
- Simulating natural phenomena
- Optimization problems

Date __/__/__
Page _____

Gene Expression Algorithm

Problem: Symbolic regression
Given a set of input-output data points, derive a mathematical expression that best models the relationship between variables using gene expression Algorithm (GEA).

Algorithm:

Step 1: Create a population of random genes where each is a sequence of terminals and functions.

Step 2: Decode each gene into an executable expression.

Step 3: Use tournament selection to pick the best gene parents

Step 4: Perform single point crossover to combine genes from 2 parents into offspring.

Step 5: Mutation: Randomly modify part of genes to introduce diversity

Step 6: Elitism carry forward the best genes from current generation to next.

Step 7: Iteration: Repeat the process

Date __/__/__
Page

for n no. of generations:

Step 8: Result: Output the best performing gene (expression) & fitness value.

Application

- Control System Optimization
- Engineering design Optimization
- Pattern Recognition and Classification

18/12/24