Index

1)	Exploration of Algorithms	8
2)	Genetic algorithm.	9
	Particle Swarm	7
3)	ant colony	8
4)	cockoo search	8
5)	Grey woolf optimization	9
6)	Parallel cellular algorithm	9
7)	optimization via gene expression	

Genetic Algorithm.

Algorithm:

1) define fitness function
```
def fitness function(x)
    return -2*x**3+3*x*x*2+12*x-5
```

2) generate initial population
```
def initialize_population(POP_size, lowerbound, upperbound)
```

3 Evaluate fitness
```
def evaluate(population):
    return np.array([fitness_function(individual) for individual in population])
```

4) select parents
```
def select_parents(population, fitness):
    min_fitness = np.min(fitness)
    if(min_fitness < 0:
        fitness += abs(min_fitness)+1
    total_fitness = np.sum(fitness)
    selection_probabilities = fitness/total_fitness
    return np.random.choice(population, size=2, p=selection_probabilities)
```

5 function to cross over parents.
```
alpha = random.random()
offspring1 = alpha * parents1 + (1-alpha) * parent2.
offspring2 = alpha * parents2 + (1-alpha) * parents1
return offspring1, offspring2.
```

4 mutation function
```
if random.random() < mutation rate
    return np.random.uniform(lower, upper_bound)
return individual.
```

```
def genetic_algorithm(pop_size, lower_bound, upper_b
    # Initialize population
    population = initialize_population(pop_size, lower_
        bound, upper_bound)

    best_solution = None
    best_fitness = -1
    for generation in range(generations):
        fitness = evaluate_fitness(population)

        current_best_fitness = np.max(fitness)
        if current_best_fitness > best_fitne:
            best_fitness = current_best_fitness
            best_soln = population[np.argmax(fi

        new_population = []

        while(len(new_population) < pop_size:
            parent1, parent2 = select_parents(population, fit

            offspring1, offspring2 = crossover(parent1, pa

            offspring1 = mutate()
            offspring2 = mutate()
            new_population.extend([offspring1, off
        population = np.array(new_population)[:pop
    return best solution
```

output: x = -2.994 (Expected -3)
f(x) = 39.6 (40)

Implemented
24/10/24

Lab-2

Particle Swarm optimization for function optimization.

Algorithm

1) define function which is needed to optimize

2) initialize parameters :-
P -> no of particles c_1 -> cognitive coeff
D -> no of dimensions c_2 -> social coef
T -> no of iterations w -> inertia weight

3) initialize particle
Particle position -> randomly initialize
Particle velocity -> randomly initialize in range

4) determine global best
gbest -> particle with best fitness
set gbest to the position of the particle with best fitness.

5) in loop from 1 to t
-> update velocity : generate r_1, r_2
$c = c_1 * r_1 * (P_{best} - x_i)$
$s = c_2 * r_2 * (g_{best} - x_i)$
$v_i^0 = w * v_i + c$
update position : $x_i = x_i + v_i^0$
calculate fitness of new position x_i
update Pbest if new fitness is greater & also gbest

6) print soln

Lab-3

Ant colony optimization for Travelling salesman Problem

Algorithm:

1) Represent the cities as nodes in a graph and construct a distance matrix

2) Initialize Parameters:
   m → no of ants
   α → importance of pheromone
   β → heuristic information
   P → pheromone evaporation rate
   Q → " deposit constant

3) For each ant:
   → Randomly select start city
   → build a complete tour by iteratively selecting next city
      ↳ for each unvisited city j, calculate probability of moving from i to j
      ↳ use the probability to select next city
   → add selected city to tour & mark as visited.

4) Calculate total length of each tour & keep track of best tour found so far

5 Update Pheromones:
   Pheromone *= (1-P)
   Pheromone_increase = Q/tour_length

6 Return the shortest tour & its length as the best solution.

Lab-4

Algorithm:

1) Define the objective function $f(x)$ to optimize & bounds of search space x_{min}, x_{max} $f(x) = x^2$

2) Initialize Parameters:
n → no of nests P_a → discovery probability
max no of iterations

3) Generate an initial population of nests with random position within search space.

4) Evaluate fitness / offers of each nest using the objective function.

5) Generate (new) soln
→ for each nest, generate new soln
$x_{new} = x_{curr} + \text{step size} \times \text{Levy flight}$
→ levy flight is a random walk with step size from a levy distribution

6) repeat steps 4-6 for specified no of iterations

7) return the nest with best fitness & corresponding position

Lab - 5

Grey wolf optimization Algorithm.

1) Define the objective function $f(x)$ & Specify bounds of search space x_{min} x_{max}

2) Initialize Parameters:
   n → no of wolves in a pack
   max iteration

3) Generate initial population of wolves with random positions

4) Evaluate fitness:
   → calculate fitness of each wolf using objective function
   → identify the best wolves: α, β, γ

5) for each wolf:

$$\vec{D}_\alpha = |\vec{C}_1 \cdot \vec{X}_\alpha - \vec{X}|$$

$$\vec{X}_1 = \vec{X}_\alpha - \vec{A}_1 \cdot \vec{D}_\alpha$$

similarly for β & γ

$$\vec{X}(t+1) = \frac{\vec{X}_1 + \vec{X}_2 + \vec{X}_3}{3}$$

$\vec{A}$ & $\vec{C}$ are coefficient vectors

6) Return the position of α wolf & its fitness

2/12/24

Lab 6

Algorithm Parallel cellular Algorithm

1) objective function : $fx = \sum x_i^2$

2) Initialization of parameters
no of cells, grid size, dimensions, bounds, no of iterations

3) initialize population with initial pos of all cells

4) Evaluate fitness for each cell & put it into 1D array

5) Identify neighbours more neighborhood structure

6) Update each cell state by copying position of its best neighbour.

7) Print best solution and its fitness

Lab 7

Optimization via Gene expansion Algorithm

Algorithm

1) Define objective function

$f(x) = \sum x_i^2$

2) Initialize parameters:
no of Genes, bounds, mutation rate
Crossover rate, no of generation

3) Generate Population P with a Gene each x_i^j → jth generation of ith individual

4) evaluate fitness using objective function $f(x)$

5) filter the Population with lower fitness value

6) Choose two parent at a time and crossover

7) Introduce variety of offsprings by randomly altering the gene

8) combine offspring to new Population

9) output the genetic sequence with best fitness