INDEX

NAME : YASHRAJ SINHA SUB : BIO-INSPIRED SYSTEM

STD : DIV : ROLL No: 1BM22CS335

SCHOOL / COLLEGE :

PAPER ACE

10/24

Date / / /

Q) What is - Genetic Algorithm?

→ A genetic algorithm (GA) is a search heuristic inspired by the principle of natural selection and genetics. It is used to solve optimization and search problems by mimicking the process of evolution. The basic idea is to evolve a population of candidates solutions to find the best solution to a given problem.

Key Components of Genetic Algorithm:

i) Population → A set of potential solution to the problem.

ii) Chromosome → A representation of a solution, typically encoded as a string of bits, numbers or characters.

iii) Fitness Function → A function that evaluates how good a solution is, typically mapping the chromosome to a numeric value representing its quality.

iv) Selection → The process of choosing the best candidate from the population based on their fitness scores.

v) Crossover → A genetic operator that combine two parent solution to create offspring solutions.

vi) Mutation → A genetic operator that introduces random changes to offspring to maintain genetic diversity.

Date / / /

vii) Replacement ⇒ The method used to determine how the new generation of solutions replaces the old one.

STEPS IN GENETIC ALGORITHM

1) Initialization → Generate an initial population of candidates solutions randomly.

2) Evaluation → Compute the fitness of each candidate

3) Selection → Choose the fittest candidate to reproduce

4) Crossover → Combine pairs of candidates to create new offspring

5) Mutation → Randomly alter some offspring ⇒ maintain diversity.

6) Replacement → Form a new generation and repeat the evaluation.

APPLICATION OF GENETIC ALGORITHM

1) Optimization problem

- Travelling salesman problem
- Knapsack problem.

2) Machine learning

- Feature selection
- Hyperparameters tuning

Date / / /

3) Robotics

- Path planning
- Control system optimization

4) Engineering

- Structural optimization
- Circuit design

5) Economics

- Portfolio Optimization
- Game theory

6) Biology

- Gene sequencing
- Protein structure prediction

7) Art and Design

- Evolutionary art
- Design optimization

8) Scheduling

- Job scheduling
- Resource allocation

24/10/24

LAB - 2

Date / / /

2) PARTICLE SWARM OPTIMIZATION FOR FUNCTION OPTIMIZATION :

i) Define the problem : We will create a mathematical function to optimize. For eg. $f(x) = x^2$. The goal is to minimize the function, which is known to be at $x = 0$

ii) Initialize parameters :

- No. of particles :> Choose an appropriate no. of particles for the swarm. Eg : num_particles = 30
- Inertia weight -> Eg : 0.5
- Cognitive co-efficient -> Eg : 1.5
- Social co-efficient -> Eg : 1.5

iii) Initialize particles -> Generate initial population

iv) Evaluate fitness.

Date / / /

v) Algorithm

(a) Define the fitness function $f(x)$.

(b) Initialise parameters:

→ Set num_particles, num_iterations, inertia_weight, cognitive_coefficient, social_coefficient

(c) Initialise particles:

→ Create an array of particles
→ For each particle:
  ↳ Randomly initialise position
  ↳ Randomly initialise velocity
  ↳ Set best_position to position
  ↳ Evaluate and store_fitness

(d) Evaluate fitness for each particle

(e) Update velocities and positions for each particle based on personal and global bests.

(f) Repeat step 4 and step 5 for num_iterations.

(g) Output the best solution found during the iterations.

vi) Pseudo code

function PSO (fitness_function , num_particles, inertia_weight , cognitive_coefficient , social_coefficient) :

Initialise particles
Create an array of particles with num_particles
For i from 0 to num_particles - 1 :
particle[i].position = Randomly initialize position
particle[i].velocity = Randomly initialize velocity
particle[i].best_position = particle[i].position
particle[i].best_fitness = fitness_function (particle[i].position)

global_best_position = particle[0].best_position
global_best_fitness = particle[0].best_fitness

For i from 0 to num_iterations :
For each particle in particles :
current_fitness = fitness_function (particle.position)

If current_fitness < particle.best_fitness :
particle.best_position = particle.position
particle.best_fitness = current_fitness

r1 = Random value in range [0,1]
r2 = Random value in range [0,1]
particle.velocity = (inertia_weight * particle velocity + cognitive_coefficient * r1 * particle best_position - particle.position) + social_coefficient * r2

* (global_best_position - particle.position))

particle.position = particle.position + particle.velocity

return global_best_position, global_best_fitness

Lab.11 22/10

Output

Begin particle swarm optimization on sphere function
Goal is to minize the function in 3 variables
Setting num_particle = 50
Setting max_iter = 100

Starting PSO algorithm

Iter = 10, best fitness = 0.189
Iter = 20, best fitness = 0.012
Iter = 30, best fitness = 0.001
Iter = 40, best fitness = 0.000
Iter = 50, best fitness = 0.000
Iter = 60, best fitness = 0.000
Iter = 70, best fitness = 0.000
Iter = 80, best fitness = 0.000
Iter = 90, best fitness = 0.000
PSO completed.
Best solution found:
['0.000004', '-0.000001', '0.000007']
fitness of best solution = 0.000000.

7/11/24

Lab - 3

Date / / /

ANT COLONY OPTIMIZATION FOR TRAVELLING SALESMAN PROBLEM (TSP)

Ant colony optimization (ACO) is a nature-inspired optimization algorithm that simulates the foraging behaviour of ant to solve combinatorial optimization problem, including TSP.
In this approach, artificial ants explore the solution space by constructing paths through cities and they communicate indirectly by depositing pheromones along the path they take. Over time, paths will higher pheromone levels are more likely to be selected by other ants.

The objective is to apply ACO to the TSP in order to find the shortest route that visit each city exactly once and return to the origin, using following steps :

i) Initialization → Define a set of cities and the distances between them. Set parameters for the ACO algorithm, including the no. of ants, pheromone influence and pheromone evaporation rate.

ii) Ant Path Construction → Each ant probabilistically constructs a path by selecting cities based on pheromone levels and distances. The probability of choosing a city is influenced by both pheromone intensity and distance.

Date / / /

iii) Pheromone update → After all ants complete their tours, pheromone is updated. Shorter paths receive more pheromone while all path experience pheromone evaporation over time.

iv) Iterative Optimization → Path construction and pheromone updates are repeated for a set number of iterations or until convergence. Over time, ants move towards the optimal solution.

v) Output → The best path found by any ant, along with the corresponding shortest distance is returned as the solution of TSP.

~~Pseudo code~~ Algorithm

Initialize pheromone matrix τ and distance matrix D
Set parameters : α, β, ρ, max-iterations, number of ants M

For each iteration :
    For each ant :
        Initialize a path starting from random city
        Repeat until all cities are visited :
            Select the next city based on pheromone and add distance (using α and β).
    End For

Evaluate the total distance of each path

- Update pheromone matrix by evaporating old pheromones and adding new pheromones.
- Track the shortest path so far.

Return the best path and its distance.

Output

Iteration 1: Best distance = 432.14904952
Iteration 2: Best distance = 432.14904921
Iteration 3: Best distance = 432.14949

⋮

Iteration 100: Best distance = 375.08122837

Best solution found: [5, 16, 1, 11, 10, 18, 9, 17, 6, 13, 7, 12, 3, 19, 4, 14, 8, 15, 2, 0]

Best distance = 375.08122837

14/11/24 Lab-4 Date / / /

CUCKOO SEARCH FOR SMART TRAFFIC CONTROL

* Algorithm :

1) Initialize the population of nests (solution) randomly within the search space

2) Evaluate the fitness of each nest based on optimization function.

3) Set the no. of nest (N), the probability of discovery (p) and the maximum no. of iteration (maxIter).

4) While (stopping condition not met and iteration < maxIter):
   Generate a new solution (nest) for each cuckoo using Levy flights:
   - For each nest i, generate a new candidate solution using Levy flight:
     new_solution_i = current_solution_i + levy_flight()

   Evaluate the fitness of the new solution
   If new solution is better, then the ~~old~~ current solution then replace the old nest with the new one.

   Abandon the worst nests (solutions)
5) Return The best solution
6) End.

* Parameter

1) The number of nests (solution)
2) The probability of discovering a cuckoo's (discovery probability)
3) The no. of iteration

Code: B
14/11/24

* Objective

To minimize the total waiting time for vehicles at the intersection.

* Parameters

i) Signal timing parameter
ii) Traffic flow ~~parameter~~
iii) Optimization function
iv) Operational constraints
v) External factors.
vi) Performance metric

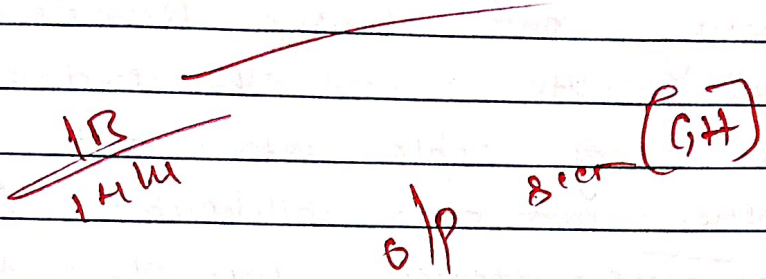
Implementation done on traffic flow management

Date / / /

Output

Best traffic signal timings: [44.769049
21.820458 64.6665304

Minimum waiting time : 8.250147 0364



21/11/24

Lab - 5

Date / /

GREY WOLF OPTIMIZER ALGORITHM FOR SOLAR PANEL PLACEMENT

The grey wolf optimizer algorithm is a swarm intelligence algorithm inspired by the social hierarchy and hunting behaviour of grey wolves. It mimics the leadership structure of alpha, beta, delta and omega wolves and their collaborative hunting strategies. This algorithm uses the social hierarchies to model the optimization process where the alpha wolves guide the search process while the beta & delta wolves assist in refining the search direction.

Algorithm

i) We are optimizing the tilt angle and orientation angle of the solar panel to maximize energy output. (Objective)

ii) Parameters => Set the no. of wolves and iterations.

iii) Randomly generate initial positions for each wolf in the search space for tilt and orientation angles.

iv) Calculate the energy output ~~and~~ according to the fitness function

v) ~~Evaluate~~ Adjust each wolves position based on alpha, beta and delta wolves.

vi) Repeat the evaluation & positions based on no. of iterations.

vii) Track and print the best position found and the corresponding energy output.

Output

Iteration 1/50 , Best Solar Energy = 1.34159827
Iteration 2/50 , Best Solar Energy = 1.32347351
⋮
Iteration 49/50 , Best Solar Energy = 1.99730324160
Iteration 50/50 , Best Solar Energy = 1.99730324165

Best placement (Tilt angle, Orientation angle) : [2041.249047 1469.3339264]

Best Solar Energy Output : 1.99730324165

28/11/24

Lab-6

Date / / /

PARALLEL CELLULAR ALGORITHM FOR TRAFFIC FLOW OPTIMIZATION

Parallel cellular algorithms are a type of algorithm inspired by how biological cell work. Just like how cells in a body work together and communicate to perform tasks, these algorithms use many small cells (which are units of the algorithm) that work together to solve problems. Each cell represents a possible solution to the problem, and by interacting with neighbours the solution is improved.

We need to model a traffic network and traffic flow in real-time. The goal is to minimize travel time, reduce congestion and optimize traffic signal timings.

Initial Parameters

- No. of cells
- Neighbourhood structure
- No. of iterations.

Date / / /

Algorithm

1) Initialize the traffic grid.

2) Randomly assign traffic ~~dest~~ densities to each intersections.

3) Define fitness function to evaluate total congestion

4) For each iteration:

   (a) Evaluate the fitness of the current grid.
   (b) For each iteration, update its state based on the traffic density.
   (c) Update grid in parallel
   (d) Track the best configuration so far.

5) Repeat the process for a set of number of iterations or until the best coverage.

6) Output the best grid and corresponding fitness value.

Date / / /

Output

Best traffic flow grid:

```
[[2 0 1 0 2 5 2 0 3 5]
 [3 1 2 0 0 0 0 2 1 0]
 [1 4 1 0 2 1 4 5 2 4]
 [0 2 5 2 2 2 2 0 5 0]
 [0 0 1 5 5 2 0 4 1 3]
 [3 0 2 3 0 1 0 5 2 4]
 [1 1 0 0 1 0 3 0 0 5]
 [1 2 4 3 5 0 5 5 5 5]
 [5 2 5 5 3 0 0 4 1 0]
 [4 5 2 2 0 3 2 1 0 0]]
```

Best Fitness (Total Congestion): 196

13/12/24 Lab-07 Date / / /

GENE EXPRESSION ALGORITHM FOR RESOURCE ALLOCATION IN BUSINESS

Gene expression algorithms (GEA) are optimization techniques inspired by the biological process of gene expression, where DNA information is converted into functional proteins. In GEA, potential solutions to a problem are treated like genetic sequences. These sequences evolve through steps like selection, crossover, mutation and gene expression to identify the best solution. GEA is widely used to solve complex optimization problems in areas like engineering, data analysis and machine learning.

Initial Parameters

Population size
Number of genes
Mutation rate
Crossover rate
No. of generations.

Algorithm

i) Define the problem.

ii) Initialize parameters => Set population size, mutation rate, crossover rate and no. of generation.

iii) Initialize population => Generate random allocations as "chromosome".

iv) Evaluate fitness => Calculate the fitness score of each chromosome based on objective function.

Date / / /

v) Selection => Select the best chromosomes (solutions) based on fitness scores.

vi) Crossover => Combine parts of two parent chromosomes to produce offspring.

vii) Mutation => Introduce small changes in the offspring to explore new solutions.

viii) Gene expression => Convert the genetic representation of the resource allocation into a

ix) Iterate => Repeat the selection, crossover, mutation and evaluation steps for several generations.

x) Output the best solution with the highest fitness.

Output

Profit per resource for each project : [8, 5, 6, 7, 4]
Generation 1, Best Profit = 700
Generation 2, Best Profit = 740
⋮
Generation 50, Best Profit = 820

Best Resource Allocation : [20, 15, 25, 30, 10]
Maximum Profit Achieved : 820