**SAPTARSHI MANDAL**

**ROLL: 2024PGCSCS09**

**EC ASSIGNMENT 5**

Q1. PSO Sphere Function

Code:  
function PSO\_sphere()

% Parameters

nVars = 4; % Number of variables

nParticles = 50; % Number of particles

maxIter = 100; % Maximum iterations

w = 0.729; % Inertia weight

c1 = 1.49445; % Cognitive coefficient

c2 = 1.49445; % Social coefficient

% Variable bounds

lowerBound = -10 \* ones(1, nVars);

upperBound = 10 \* ones(1, nVars);

% Initialize particles

particles = repmat(lowerBound, nParticles, 1) + ...

repmat(upperBound - lowerBound, nParticles, 1) .\* rand(nParticles, nVars);

% Initialize velocities

velocities = zeros(nParticles, nVars);

% Initialize personal best

personalBest = particles;

personalBestFitness = inf(nParticles, 1);

% Initialize global best

globalBest = zeros(1, nVars);

globalBestFitness = inf;

% Fitness history

fitnessHistory = zeros(maxIter, 1);

% Evaluate initial population

for i = 1:nParticles

currentFitness = sphereFunction(particles(i,:));

personalBestFitness(i) = currentFitness;

if currentFitness < globalBestFitness

globalBestFitness = currentFitness;

globalBest = particles(i,:);

end

end

% Main PSO loop

for iter = 1:maxIter

for i = 1:nParticles

% Update velocity

r1 = rand(1, nVars);

r2 = rand(1, nVars);

cognitive = c1 \* r1 .\* (personalBest(i,:) - particles(i,:));

social = c2 \* r2 .\* (globalBest - particles(i,:));

velocities(i,:) = w \* velocities(i,:) + cognitive + social;

% Update position

particles(i,:) = particles(i,:) + velocities(i,:);

% Apply bounds

particles(i,:) = max(particles(i,:), lowerBound);

particles(i,:) = min(particles(i,:), upperBound);

% Evaluate fitness

currentFitness = sphereFunction(particles(i,:));

% Update personal best

if currentFitness < personalBestFitness(i)

personalBestFitness(i) = currentFitness;

personalBest(i,:) = particles(i,:);

% Update global best

if currentFitness < globalBestFitness

globalBestFitness = currentFitness;

globalBest = particles(i,:);

end

end

end

% Store best fitness

fitnessHistory(iter) = globalBestFitness;

% Display progress

if mod(iter, 10) == 0

fprintf('Iteration %d: Best fitness = %f\n', iter, globalBestFitness);

end

end

% Results

fprintf('\nOptimization completed:\n');

fprintf('Best solution found: [%f, %f, %f, %f]\n', globalBest);

fprintf('Minimum function value: %f\n', globalBestFitness);

% Plot fitness history

figure;

plot(1:maxIter, fitnessHistory, 'LineWidth', 2);

xlabel('Iteration');

ylabel('Best Fitness');

title('Convergence of PSO on Sphere Function');

grid on;

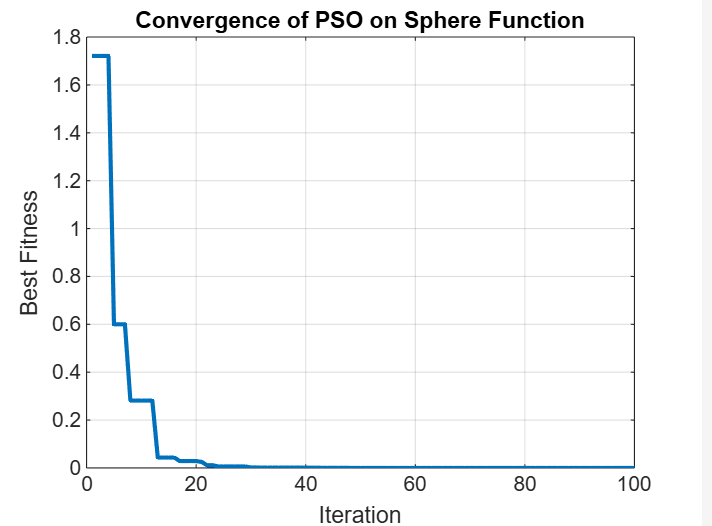
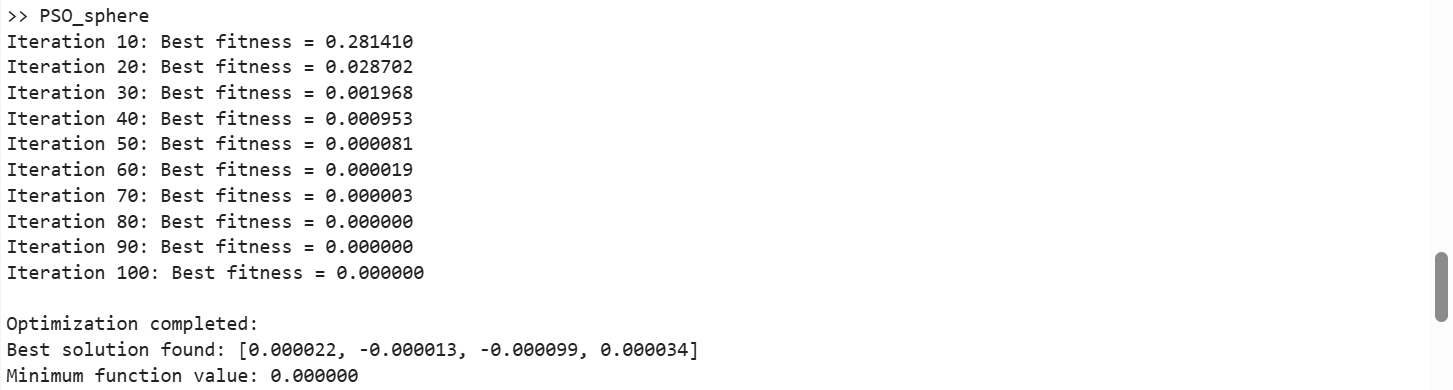
end

% Sphere function

function f = sphereFunction(x)

f = sum(x.^2);

end

OUTPUT:  


Q2. GAP Problem using PSO

Code:  
clc; clear; close all;

% List of GAP files

numFiles = 12;

filePrefix = 'gap'; % Files are named gap1.txt, gap2.txt, ..., gap12.txt

% PSO Parameters

numParticles = 50; % Swarm size

numIterations = 100; % Maximum iterations

w = 0.7; % Inertia weight

c1 = 1.5; % Cognitive coefficient

c2 = 1.5; % Social coefficient

% Store results for formatted output

results = cell(numFiles, 1);

headers = strings(1, numFiles);

% Open file to write results

outputFile = fopen('results\_pso.txt', 'w');

fprintf(outputFile, 'InstanceID,Profit\n');

for fileIdx = 1:numFiles

filename = sprintf('%s%d.txt', filePrefix, fileIdx);

fileID = fopen(filename, 'r');

if fileID == -1

fprintf('Error: Unable to open %s\n', filename);

continue;

end

% Read number of problems in the file

numProblems = fscanf(fileID, '%d', 1);

problemResults = strings(numProblems, 1);

% Extract filename without extension for header

[~, baseName, ~] = fileparts(filename);

headers(fileIdx) = sprintf('%-20s', baseName); % Header formatting

for p = 1:numProblems

% Read problem parameters

numServers = fscanf(fileID, '%d', 1);

numUsers = fscanf(fileID, '%d', 1);

% Read utility values

U = fscanf(fileID, '%d', [numUsers, numServers])';

% Read resource requirement matrix

R = fscanf(fileID, '%d', [numUsers, numServers])';

% Read server capacity

capacity = fscanf(fileID, '%d', numServers);

% Initialize PSO swarm

position = rand(numParticles, numServers \* numUsers); % Random initialization

velocity = zeros(numParticles, numServers \* numUsers);

% Best positions & fitness values

pBest = position;

pBestFitness = -inf(numParticles, 1);

[gBestFitness, gBestIdx] = max(pBestFitness);

gBest = pBest(gBestIdx, :);

% PSO Main Loop

for iter = 1:numIterations

fitness = zeros(numParticles, 1);

for i = 1:numParticles

% Convert continuous position values into binary assignment

xBinary = reshape(position(i, :), numServers, numUsers);

xBinary = (xBinary == max(xBinary)); % Assign user to the best server

% Constraint check

if all(sum(xBinary, 1) == 1) && all(all(sum(R .\* xBinary, 2) <= capacity'))

fitness(i) = sum(sum(U .\* xBinary)); % Valid solution, calculate utility

else

fitness(i) = -inf; % Penalize infeasible solutions

end

end

% Update personal & global bests

betterIdx = fitness > pBestFitness;

pBest(betterIdx, :) = position(betterIdx, :);

pBestFitness(betterIdx) = fitness(betterIdx);

[newGBestFitness, newGBestIdx] = max(pBestFitness);

if newGBestFitness > gBestFitness

gBestFitness = newGBestFitness;

gBest = pBest(newGBestIdx, :);

end

% Update velocity & position

velocity = w \* velocity ...

+ c1 \* rand(numParticles, numServers \* numUsers) .\* (pBest - position) ...

+ c2 \* rand(numParticles, numServers \* numUsers) .\* (gBest - position);

position = position + velocity;

% Clamp positions between [0,1]

position = max(0, min(1, position));

end

% Best solution for this problem instance

bestAssignment = reshape(gBest, numServers, numUsers);

bestAssignment = (bestAssignment == max(bestAssignment)); % Convert to valid binary form

bestUtility = sum(sum(U .\* bestAssignment));

% Format output correctly with spacing

problemID = sprintf('c%d%d-%d', numServers, numUsers, p);

problemResults(p) = sprintf('%-12s %-6d', problemID, bestUtility);

% Write to results\_pso.txt

fprintf(outputFile, '%s,%d\n', problemID, bestUtility);

end

results{fileIdx} = problemResults;

fclose(fileID);

end

% Close the output file

fclose(outputFile);

% Print formatted output in groups of 4 files per row

colsPerRow = 4;

numRows = ceil(numFiles / colsPerRow);

for row = 1:numRows

colStart = (row - 1) \* colsPerRow + 1;

colEnd = min(row \* colsPerRow, numFiles);

% Print headers

fprintf('\n');

for col = colStart:colEnd

fprintf('%-22s', headers(col));

end

fprintf('\n');

% Print problem results line by line

maxProblems = max(cellfun(@numel, results(colStart:colEnd)));

for p = 1:maxProblems

for col = colStart:colEnd

if p <= numel(results{col})

fprintf('%-22s', results{col}(p));

else

fprintf('%-22s', ''); % Empty space for alignment

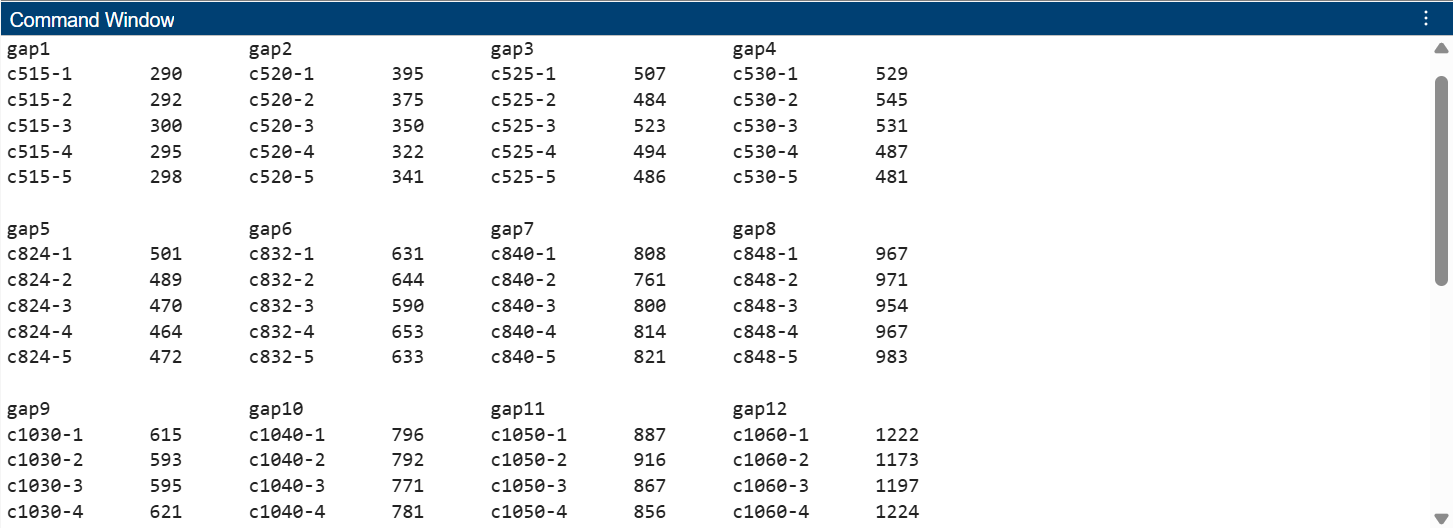
end

end

fprintf('\n');

end

end

OUTPUT:  
  
  
  
Results Compare Code:  
function convergence\_comparison()

% Load data from files

optimalData = readtable('results\_optimal.txt', 'Delimiter', ',', 'VariableNamingRule', 'preserve');

approxData = readtable('results\_approx.txt', 'Delimiter', ',', 'VariableNamingRule', 'preserve');

gaBinaryData = readtable('results\_ga\_binary.txt', 'Delimiter', ',', 'VariableNamingRule', 'preserve');

gaRealData = readtable('results\_ga\_real.txt', 'Delimiter', ',', 'VariableNamingRule', 'preserve'); % Load the new file

psoData = readtable('results\_pso.txt', 'Delimiter', ',', 'VariableNamingRule', 'preserve'); % Load the PSO file

% Extract Instance IDs and values

optimalIDs = optimalData.InstanceID;

optimalValues = optimalData.OptimalCost;

approxIDs = approxData.InstanceID;

approxValues = approxData.Profit;

gaBinaryIDs = gaBinaryData.InstanceID;

gaBinaryValues = gaBinaryData.Profit;

gaRealIDs = gaRealData.InstanceID;

gaRealValues = gaRealData.Profit; % Get profit from the real GA data

psoIDs = psoData.InstanceID;

psoValues = psoData.Profit; % Get profit from PSO data

% Match common IDs among all datasets

[commonIDs12, idxOptimal, idxApprox] = intersect(optimalIDs, approxIDs, 'stable');

[commonIDs, idx12, idxGA] = intersect(commonIDs12, gaBinaryIDs, 'stable');

[commonIDsFinal, idxGAReal] = intersect(commonIDs, gaRealIDs, 'stable'); % Match with gaReal

[commonIDsFinal2, idxPSO] = intersect(commonIDsFinal, psoIDs, 'stable'); % Match with PSO

% Final matching indices

matchedOptimal = optimalValues(idxOptimal(idx12));

matchedApprox = approxValues(idxApprox(idx12));

matchedGA = gaBinaryValues(idxGA);

matchedGAReal = gaRealValues(idxGAReal); % Matched real GA results

matchedPSO = psoValues(idxPSO); % Matched PSO results

% Plot comparison

figure;

plot(1:length(commonIDsFinal2), matchedOptimal, '-o', 'LineWidth', 2);

hold on;

plot(1:length(commonIDsFinal2), matchedApprox, '-x', 'LineWidth', 2);

plot(1:length(commonIDsFinal2), matchedGA, '-s', 'LineWidth', 2);

plot(1:length(commonIDsFinal2), matchedGAReal, '-^', 'LineWidth', 2); % Add real GA to the plot

plot(1:length(commonIDsFinal2), matchedPSO, '-d', 'LineWidth', 2); % Add PSO to the plot

xlabel('Instance Index');

ylabel('Profit');

title('Optimal vs Approximate vs GA Binary vs GA Real vs PSO Profit Comparison');

legend('Optimal', 'Approximate', 'GA (Binary)', 'GA (Real)', 'PSO','Location','northwest');

grid on;

end

OUTPUT:  
