**SAPTARSHI MANDAL**

**2024PGCSCS09**

**EC ASSIGNMENT 6**

Q1. TLBO Sphere Function

Code:  
function TLBO\_sphere\_improved()

% Parameters

nVars = 4; % Number of variables

popSize = 50; % Population size (class size)

maxIter = 200; % Maximum iterations (increased for better exploration)

tol = 1e-6; % Tolerance for convergence checking

% Variable bounds

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

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

% Initialize population with better diversity

population = repmat(lowerBound, popSize, 1) + ...

repmat(upperBound - lowerBound, popSize, 1) .\* lhsdesign(popSize, nVars, 'iterations', 1000);

% Evaluate initial population

fitness = zeros(popSize, 1);

for i = 1:popSize

fitness(i) = sphereFunction(population(i,:));

end

% Initialize best solution tracking

[bestFitness, bestIdx] = min(fitness);

bestSolution = population(bestIdx,:);

fitnessHistory = zeros(maxIter, 1);

diversityHistory = zeros(maxIter, 1);

% Main TLBO loop

for iter = 1:maxIter

% Calculate population diversity

diversity = mean(std(population));

diversityHistory(iter) = diversity;

% Teacher Phase (Learning from the teacher)

[~, teacherIdx] = min(fitness);

teacher = population(teacherIdx,:);

% Calculate mean of the population

meanPopulation = mean(population, 1);

% Adaptive teaching factor

TF = 1 + rand(); % Now varies between 1 and 2

% Update each learner with momentum

newPopulation = population;

for i = 1:popSize

% Difference between teacher and mean with small random component

difference = (teacher - (TF \* meanPopulation)) .\* (0.9 + 0.2\*rand(1,nVars));

% Generate new solution with learning rate adjustment

learningRate = 0.5 \* (1 + (iter/maxIter)); % Decreases over time

newSolution = population(i,:) + learningRate \* rand(1, nVars) .\* difference;

% Apply bounds with bounce-back

outOfBounds = (newSolution < lowerBound) | (newSolution > upperBound);

newSolution(outOfBounds) = population(i,outOfBounds) - 0.5\*rand(1,sum(outOfBounds)) .\* ...

(newSolution(outOfBounds) - population(i,outOfBounds));

% Ensure we stay within bounds

newSolution = max(newSolution, lowerBound);

newSolution = min(newSolution, upperBound);

% Evaluate new solution

newFitness = sphereFunction(newSolution);

% Greedy selection with small probability to accept worse solutions

if newFitness < fitness(i) || rand() < 0.05\*(1-iter/maxIter)

newPopulation(i,:) = newSolution;

fitness(i) = newFitness;

end

end

population = newPopulation;

% Learner Phase (Learning from peers with diversity maintenance)

for i = 1:popSize

% Select partner using tournament selection for better diversity

candidates = randperm(popSize, 3);

[~, bestCandidate] = min(fitness(candidates));

partner = candidates(bestCandidate);

if partner == i

partner = candidates(mod(bestCandidate,3)+1);

end

% Adaptive learning from partner

if fitness(i) < fitness(partner)

difference = (population(i,:) - population(partner,:)) .\* (0.5 + rand(1,nVars));

else

difference = (population(partner,:) - population(i,:)) .\* (0.5 + rand(1,nVars));

end

% Generate new solution with decreasing perturbation

perturbation = 0.1 \* (maxIter - iter)/maxIter;

newSolution = population(i,:) + (rand(1,nVars)+perturbation) .\* difference;

% Apply bounds with reflection

newSolution = max(newSolution, lowerBound);

newSolution = min(newSolution, upperBound);

% Evaluate new solution

newFitness = sphereFunction(newSolution);

% Probabilistic acceptance

if newFitness < fitness(i) || rand() < exp((fitness(i)-newFitness)/diversity)

population(i,:) = newSolution;

fitness(i) = newFitness;

end

end

% Update best solution

[currentBest, idx] = min(fitness);

if currentBest < bestFitness

bestFitness = currentBest;

bestSolution = population(idx,:);

end

fitnessHistory(iter) = bestFitness;

% Display progress with diversity information

if mod(iter, 10) == 0

fprintf('Iter %4d: BestFit = %.4e, Diversity = %.4f\n', ...

iter, bestFitness, diversity);

end

% Early stopping if diversity is too low but solution not optimal

if diversity < tol && bestFitness > tol

fprintf('Restarting due to premature convergence\n');

% Reset worst half of population

[~, worstIdx] = sort(fitness, 'descend');

population(worstIdx(1:round(popSize/2)),:) = ...

repmat(lowerBound, round(popSize/2), 1) + ...

repmat(upperBound - lowerBound, round(popSize/2), 1) .\* rand(round(popSize/2), nVars);

% Re-evaluate

for i = worstIdx(1:round(popSize/2))

fitness(i) = sphereFunction(population(i,:));

end

end

% Check for true convergence

if bestFitness < tol

break;

end

end

% Results

fprintf('\nOptimization completed after %d iterations:\n', iter);

fprintf('Best solution found: [');

fprintf('%.6f, ', bestSolution(1:end-1));

fprintf('%.6f]\n', bestSolution(end));

fprintf('Minimum function value: %.12f\n', bestFitness);

fprintf('Final diversity measure: %f\n', diversity);

% Plot results

figure;

subplot(2,1,1);

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

xlabel('Iteration');

ylabel('Best Fitness');

title('Convergence of Improved TLBO on Sphere Function');

grid on;

subplot(2,1,2);

plot(1:iter, diversityHistory(1:iter), 'LineWidth', 2);

xlabel('Iteration');

ylabel('Population Diversity');

title('Population Diversity During Optimization');

grid on;

end

% Sphere function with numerical stability check

function f = sphereFunction(x)

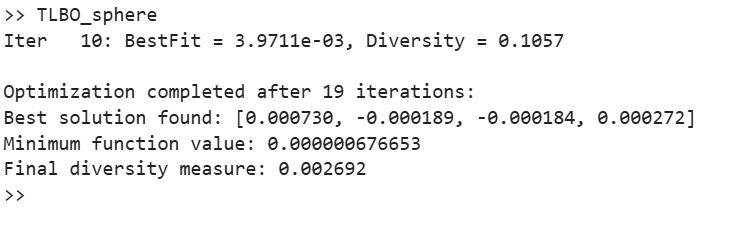
% Add small noise to prevent exact zero comparisons

perturbed\_x = x + 1e-10\*randn(size(x));

f = sum(perturbed\_x.^2);

end

Output:

  
Q2: GAP Problem using TLBO

Code:  
clc; clear; close all;

% List of GAP files

numFiles = 12;

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

% TLBO Parameters

numLearners = 50; % Population size

numIterations = 100; % Maximum iterations

% Store results for formatted output

results = cell(numFiles, 1);

headers = strings(1, numFiles);

% Open file to write results

outputFile = fopen('results\_tlbo.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);

dim = numServers \* numUsers;

% Initialize learners (solutions)

population = rand(numLearners, dim);

fitness = -inf(numLearners, 1);

% Evaluate initial fitness

for i = 1:numLearners

x = reshape(population(i, :), numServers, numUsers);

xBinary = (x == max(x));

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

fitness(i) = sum(sum(U .\* xBinary));

else

fitness(i) = -inf;

end

end

% TLBO Main Loop

for iter = 1:numIterations

% Teacher Phase

[bestFitness, teacherIdx] = max(fitness);

teacher = population(teacherIdx, :);

meanLearner = mean(population);

for i = 1:numLearners

TF = randi([1, 2]); % Teaching factor either 1 or 2

newSol = population(i, :) + rand(1, dim) .\* (teacher - TF \* meanLearner);

% Clamp to [0, 1]

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

% Evaluate new solution

x = reshape(newSol, numServers, numUsers);

xBinary = (x == max(x));

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

newFitness = sum(sum(U .\* xBinary));

else

newFitness = -inf;

end

if newFitness > fitness(i)

population(i, :) = newSol;

fitness(i) = newFitness;

end

end

% Learner Phase

for i = 1:numLearners

partnerIdx = randi(numLearners);

while partnerIdx == i

partnerIdx = randi(numLearners);

end

if fitness(i) > fitness(partnerIdx)

newSol = population(i, :) + rand(1, dim) .\* (population(i, :) - population(partnerIdx, :));

else

newSol = population(i, :) + rand(1, dim) .\* (population(partnerIdx, :) - population(i, :));

end

% Clamp to [0, 1]

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

% Evaluate new solution

x = reshape(newSol, numServers, numUsers);

xBinary = (x == max(x));

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

newFitness = sum(sum(U .\* xBinary));

else

newFitness = -inf;

end

if newFitness > fitness(i)

population(i, :) = newSol;

fitness(i) = newFitness;

end

end

end

% Best solution after TLBO

[bestFitness, bestIdx] = max(fitness);

bestAssignment = reshape(population(bestIdx, :), numServers, numUsers);

bestAssignment = (bestAssignment == max(bestAssignment));

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\_tlbo.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

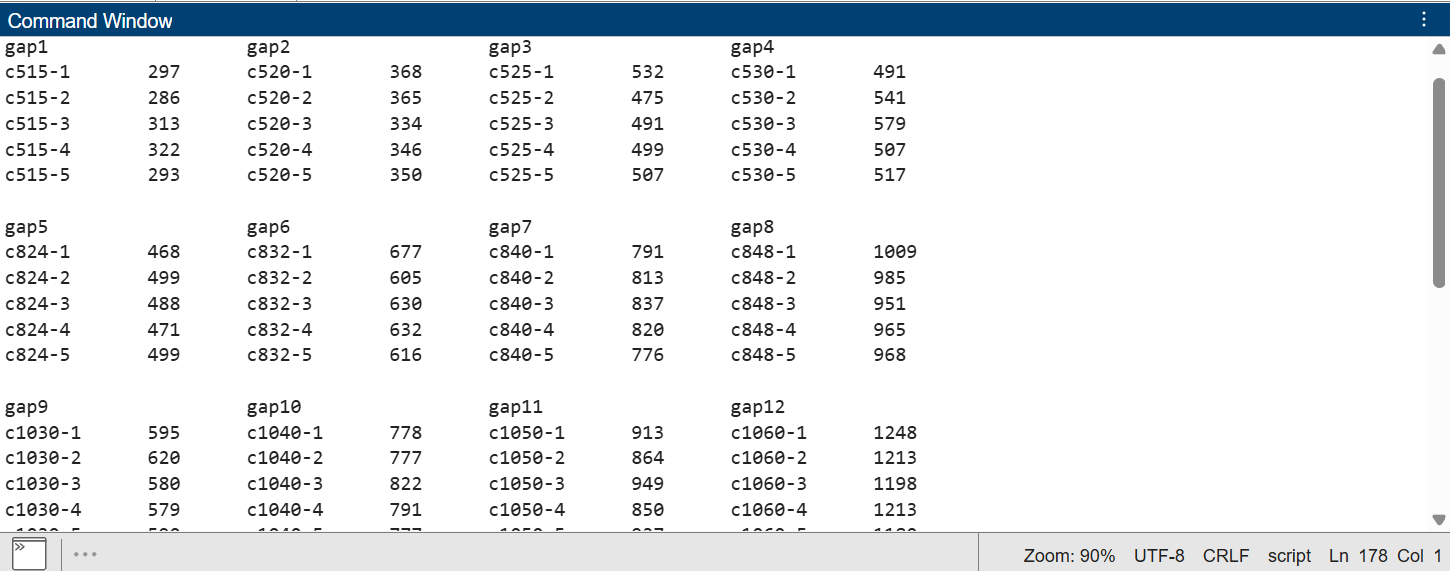
end

end

fprintf('\n');

end

end

Output:  
  
  
Comparison 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');

psoData = readtable('results\_pso.txt', 'Delimiter', ',', 'VariableNamingRule', 'preserve');

tlboData = readtable('results\_tlbo.txt', 'Delimiter', ',', 'VariableNamingRule', 'preserve'); % New line

% 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;

psoIDs = psoData.InstanceID;

psoValues = psoData.Profit;

tlboIDs = tlboData.InstanceID;

tlboValues = tlboData.Profit; % New line

% Match common IDs among all datasets step-by-step

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

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

[commonIDsFinal, idxGAReal] = intersect(commonIDs, gaRealIDs, 'stable');

[commonIDsFinal2, idxPSO] = intersect(commonIDsFinal, psoIDs, 'stable');

[commonIDsFinal3, idxTLBO] = intersect(commonIDsFinal2, tlboIDs, 'stable'); % Match with TLBO

% Final matched values

matchedOptimal = optimalValues(idxOptimal(idx12(idxGAReal(idxPSO))));

matchedApprox = approxValues(idxApprox(idx12(idxGAReal(idxPSO))));

matchedGA = gaBinaryValues(idxGA(idxGAReal(idxPSO)));

matchedGAReal = gaRealValues(idxGAReal(idxPSO));

matchedPSO = psoValues(idxPSO);

matchedTLBO = tlboValues(idxTLBO); % New line

% Plot comparison

figure;

plot(1:length(commonIDsFinal3), matchedOptimal, '-o', 'LineWidth', 2);

hold on;

plot(1:length(commonIDsFinal3), matchedApprox, '-x', 'LineWidth', 2);

plot(1:length(commonIDsFinal3), matchedGA, '-s', 'LineWidth', 2);

plot(1:length(commonIDsFinal3), matchedGAReal, '-^', 'LineWidth', 2);

plot(1:length(commonIDsFinal3), matchedPSO, '-d', 'LineWidth', 2);

plot(1:length(commonIDsFinal3), matchedTLBO, '-p', 'LineWidth', 2); % New line

xlabel('Instance Index');

ylabel('Profit');

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

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

grid on;

end

Output:  
