**SAPTARSHI MANDAL**

**2024PGCSCS09**

**EC ASSIGNMENT 4**

Q1. Real GA Sphere Function Code:  
  
function realGA\_SBX\_sphere()

% Parameters

nVars = 4; % Number of variables

popSize = 50; % Population size

maxGen = 100; % Maximum generations

pc = 0.8; % Crossover probability

pm = 0.1; % Mutation probability

eta\_c = 15; % SBX distribution index (typically 2-20)

eta\_m = 20; % Polynomial mutation distribution index

% Variable bounds

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

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

% Initialize population

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

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

% Store best fitness and solution

bestFitness = inf;

bestSolution = zeros(1, nVars);

fitnessHistory = zeros(maxGen, 1);

% Main GA loop

for gen = 1:maxGen

% Evaluate fitness

fitness = zeros(popSize, 1);

for i = 1:popSize

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

end

% Update best solution

[minFit, idx] = min(fitness);

if minFit < bestFitness

bestFitness = minFit;

bestSolution = pop(idx,:);

end

fitnessHistory(gen) = bestFitness;

% Selection (Tournament selection)

selectedIndices = tournamentSelection(fitness, popSize);

selectedPop = pop(selectedIndices, :);

% SBX Crossover

offspring = sbxCrossover(selectedPop, pc, eta\_c, lowerBound, upperBound);

% Polynomial Mutation

offspring = polynomialMutation(offspring, pm, eta\_m, lowerBound, upperBound);

% Elitism: Keep best individual

[~, bestIdx] = min(fitness);

offspring(1,:) = pop(bestIdx,:);

% Update population

pop = offspring;

% Display progress

if mod(gen, 10) == 0

fprintf('Generation %d: Best fitness = %f\n', gen, bestFitness);

end

end

% Results

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

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

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

% Plot fitness history

figure;

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

xlabel('Generation');

ylabel('Best Fitness');

title('Convergence of Real-Coded GA with SBX on Sphere Function');

grid on;

end

% Sphere function

function f = sphereFunction(x)

f = sum(x.^2);

end

% Tournament selection

function selectedIndices = tournamentSelection(fitness, popSize)

selectedIndices = zeros(popSize, 1);

tournamentSize = 2; % Tournament size

for i = 1:popSize

% Randomly select tournamentSize individuals

contestants = randperm(popSize, tournamentSize);

[~, bestIdx] = min(fitness(contestants));

selectedIndices(i) = contestants(bestIdx);

end

end

% Simulated Binary Crossover (SBX)

function offspring = sbxCrossover(parents, pc, eta\_c, lowerBound, upperBound)

[popSize, nVars] = size(parents);

offspring = parents;

for i = 1:2:popSize-1

if rand < pc

parent1 = parents(i,:);

parent2 = parents(i+1,:);

child1 = zeros(1, nVars);

child2 = zeros(1, nVars);

for j = 1:nVars

if rand <= 0.5

% Ensure parents are ordered

if abs(parent1(j) - parent2(j)) > 1e-10

if parent1(j) < parent2(j)

x1 = parent1(j);

x2 = parent2(j);

else

x1 = parent2(j);

x2 = parent1(j);

end

% Calculate beta

u = rand;

if u <= 0.5

beta = (2\*u)^(1/(eta\_c+1));

else

beta = (1/(2\*(1-u)))^(1/(eta\_c+1));

end

% Create children

child1(j) = 0.5 \* ((x1 + x2) - beta\*(x2 - x1));

child2(j) = 0.5 \* ((x1 + x2) + beta\*(x2 - x1));

% Ensure children are within bounds

child1(j) = max(min(child1(j), upperBound(j)), lowerBound(j));

child2(j) = max(min(child2(j), upperBound(j)), lowerBound(j));

else

child1(j) = parent1(j);

child2(j) = parent2(j);

end

else

child1(j) = parent1(j);

child2(j) = parent2(j);

end

end

offspring(i,:) = child1;

offspring(i+1,:) = child2;

end

end

end

% Polynomial Mutation

function offspring = polynomialMutation(offspring, pm, eta\_m, lowerBound, upperBound)

[popSize, nVars] = size(offspring);

for i = 1:popSize

for j = 1:nVars

if rand < pm

y = offspring(i,j);

yl = lowerBound(j);

yu = upperBound(j);

delta1 = (y - yl) / (yu - yl);

delta2 = (yu - y) / (yu - yl);

r = rand;

if r <= 0.5

deltaq = (2\*r + (1-2\*r)\*(1-delta1)^(eta\_m+1))^(1/(eta\_m+1)) - 1;

else

deltaq = 1 - (2\*(1-r) + 2\*(r-0.5)\*(1-delta2)^(eta\_m+1))^(1/(eta\_m+1));

end

y = y + deltaq \* (yu - yl);

y = min(max(y, yl), yu);

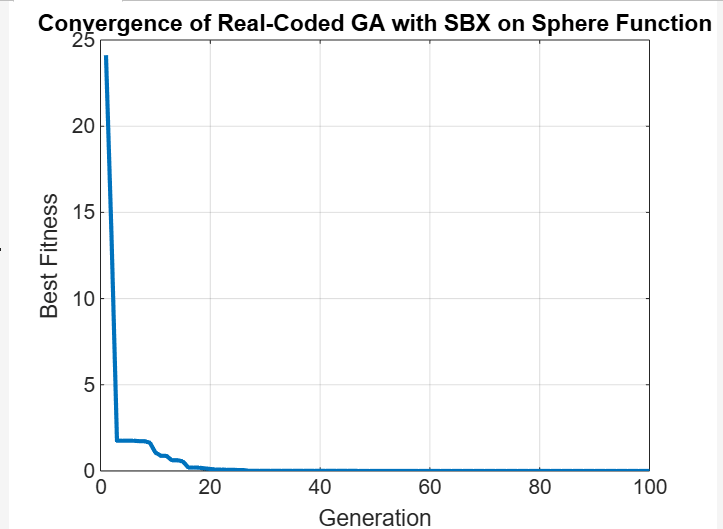
offspring(i,j) = y;

end

end

end

end

Output:  
  
  


Q2. GAP PROBLEM REAL CODED GA

function solve\_large\_gap\_ga()

results = {}; % Cell array to store results

% Iterate through gap1 to gap12 dataset files

for g = 1:12

filename = sprintf('gap%d.txt', g);

fid = fopen(filename, 'r');

if fid == -1

error('Error opening file %s.', filename);

end

% Read the number of problem sets

num\_problems = fscanf(fid, '%d', 1);

% Print dataset name (gapX)

fprintf('\n%s\n', filename(1:end-4)); % Removes .txt for display

for p = 1:num\_problems

m = fscanf(fid, '%d', 1); % Number of servers

n = fscanf(fid, '%d', 1); % Number of users

c = fscanf(fid, '%d', [n, m])';

r = fscanf(fid, '%d', [n, m])';

b = fscanf(fid, '%d', [m, 1]);

is\_gap12 = (g == 12); % For convergence plot

[x\_matrix, fitness\_log] = solve\_gap\_ga(m, n, c, r, b, is\_gap12);

objective\_value = sum(sum(c .\* x\_matrix)); % Maximization

instance\_id = sprintf('c%d-%d', m\*100 + n, p);

fprintf('%s %d\n', instance\_id, round(objective\_value));

results{end+1, 1} = instance\_id;

results{end, 2} = round(objective\_value);

% Plot convergence for GAP12

if is\_gap12

figure;

plot(1:length(fitness\_log), fitness\_log, '-o', 'LineWidth', 2);

xlabel('Generation');

ylabel('Best Fitness Value');

title(sprintf('Fitness Convergence for %s - Problem %d', filename(1:end-4), p));

grid on;

end

end

fclose(fid);

end

% Save to file

output\_filename = fullfile(pwd, 'results\_ga\_real.txt');

outFile = fopen(output\_filename, 'w');

if outFile == -1

error('Unable to open the output file: %s', output\_filename);

end

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

for i = 1:size(results, 1)

instance\_id = results{i, 1};

profit = results{i, 2};

fprintf(outFile, '%s,%d\n', instance\_id, profit);

end

fclose(outFile);

fprintf('Results successfully saved to %s\n', output\_filename);

end

function [x\_matrix, fitness\_log] = solve\_gap\_ga(m, n, c, r, b, plot\_convergence)

pop\_size = 100;

max\_gen = 100;

crossover\_rate = 0.8;

mutation\_rate = 0.02;

eta = 20; % Distribution index for SBX

fitness\_log = zeros(1, max\_gen);

population = rand(pop\_size, m \* n);

for i = 1:pop\_size

population(i, :) = enforce\_feasibility(rand(1, m \* n), m, n);

end

fitness = arrayfun(@(i) fitnessFcn(population(i, :)), 1:pop\_size);

for gen = 1:max\_gen

parents = tournamentSelection(population, fitness);

offspring = simulatedBinaryCrossover(parents, crossover\_rate, eta);

mutated\_offspring = mutation(offspring, mutation\_rate);

for i = 1:size(mutated\_offspring, 1)

mutated\_offspring(i, :) = enforce\_feasibility(mutated\_offspring(i, :), m, n);

end

new\_fitness = arrayfun(@(i) fitnessFcn(mutated\_offspring(i, :)), 1:size(mutated\_offspring, 1));

[~, best\_idx] = max([fitness, new\_fitness]);

if best\_idx > length(fitness)

population = mutated\_offspring;

fitness = new\_fitness;

else

population = [population; mutated\_offspring];

fitness = [fitness, new\_fitness];

end

[~, sorted\_idx] = sort(fitness, 'descend');

population = population(sorted\_idx(1:pop\_size), :);

fitness = fitness(sorted\_idx(1:pop\_size));

if plot\_convergence

fitness\_log(gen) = fitness(1); % Store best fitness of current gen

end

end

[~, best\_idx] = max(fitness);

x\_matrix = reshape(population(best\_idx, :), [m, n]);

function fval = fitnessFcn(x)

x\_mat = reshape(x, [m, n]);

cost = sum(sum(c .\* x\_mat));

capacity\_violation = sum(max(sum(x\_mat .\* r, 2) - b, 0));

assignment\_violation = sum(abs(sum(x\_mat, 1) - 1));

penalty = 1e6 \* (capacity\_violation + assignment\_violation);

fval = cost - penalty;

end

end

function selected = tournamentSelection(population, fitness)

pop\_size = size(population, 1);

selected = zeros(size(population));

for i = 1:pop\_size

idx1 = randi(pop\_size);

idx2 = randi(pop\_size);

if fitness(idx1) > fitness(idx2)

selected(i, :) = population(idx1, :);

else

selected(i, :) = population(idx2, :);

end

end

end

function offspring = simulatedBinaryCrossover(parents, crossover\_rate, eta)

pop\_size = size(parents, 1);

num\_genes = size(parents, 2);

offspring = parents;

for i = 1:2:pop\_size-1

if rand < crossover\_rate

% Select parents

parent1 = parents(i, :);

parent2 = parents(i+1, :);

% Generate offspring using SBX

for j = 1:num\_genes

u = rand;

if u <= 0.5

beta = (2 \* u)^(1 / (eta + 1));

else

beta = (1 / (2 \* (1 - u)))^(1 / (eta + 1));

end

offspring(i, j) = 0.5 \* ((1 + beta) \* parent1(j) + (1 - beta) \* parent2(j));

offspring(i+1, j) = 0.5 \* ((1 - beta) \* parent1(j) + (1 + beta) \* parent2(j));

end

end

end

end

function mutated = mutation(offspring, mutation\_rate)

mutated = offspring;

for i = 1:numel(offspring)

if rand < mutation\_rate

mutated(i) = rand; % Real-valued mutation

end

end

end

function x\_corrected = enforce\_feasibility(x, m, n)

x\_mat = reshape(x, [m, n]);

for j = 1:n

[~, idx] = max(x\_mat(:, j));

x\_mat(:, j) = 0;

x\_mat(idx, j) = 1;

end

x\_corrected = reshape(x\_mat, [1, m \* n]);

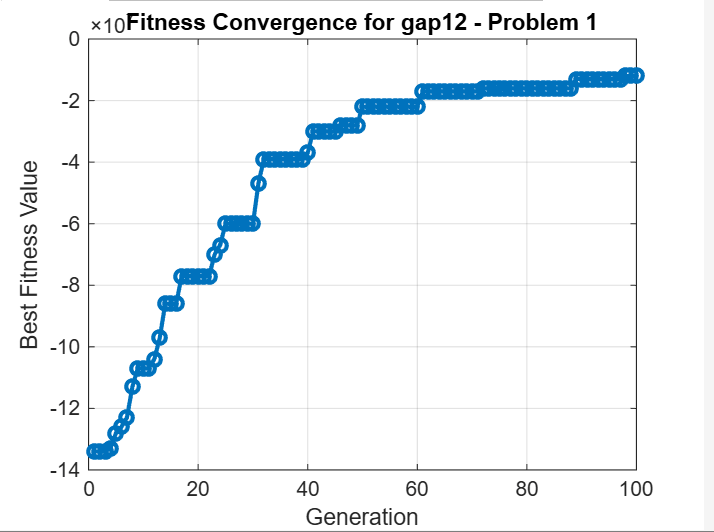
end

OUTPUT:  






GAP 12 Convergence:

  
  
  
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

% 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

% 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

% Final matching indices

matchedOptimal = optimalValues(idxOptimal(idx12));

matchedApprox = approxValues(idxApprox(idx12));

matchedGA = gaBinaryValues(idxGA);

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

% Plot comparison

figure;

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

hold on;

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

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

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

xlabel('Instance Index');

ylabel('Profit');

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

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

grid on;

end

OUTPUT:  
