Assignment :- 3 (Binary coded GA)

Name :- Sanjeevani Sahu

Roll no. :- 2024PGCSCS03

Run 20 iterations of gap 12 (instance 1) and then take avg :-

function optimize\_gap\_ga()

% Only run gap12.txt and focus on its 1st instance

dataFile = 'gap12.txt';

fileID = fopen(dataFile, 'r');

if fileID == -1

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

end

% Read number of instances in file

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

% Read only the 1st instance data

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

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

% Read cost matrix (numUsers x numServers), then transpose to [Servers x Users]

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

% Read resource matrix (numUsers x numServers), then transpose to [Servers x Users]

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

% Read server capacity limits

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

fclose(fileID); % Done reading

% Prepare result file for writing

resultFile = 'avg\_of\_gap12(instance1).txt';

resultFID = fopen(resultFile, 'w');

if resultFID == -1

error('Unable to create result file.');

end

fprintf(resultFID, 'Results for gap12 (Instance 1) using Genetic Algorithm\n');

fprintf(resultFID, '--------------------------------------------------------\n');

% Run GA for 20 iterations

totalBenefits = zeros(1, 20);

for run = 1:20

assignmentMatrix = execute\_ga(numServers, numUsers, costMatrix, resourceMatrix, capacityLimits);

totalBenefits(run) = sum(sum(costMatrix .\* assignmentMatrix)); % Maximization

fprintf('Run %2d: Total Benefit = %d\n', run, round(totalBenefits(run)));

fprintf(resultFID, 'Run %2d: Total Benefit = %d\n', run, round(totalBenefits(run)));

end

% Display average total benefit

avgBenefit = mean(totalBenefits);

fprintf('\nAverage Total Benefit after 20 runs on gap12 (instance 1): %.2f\n', avgBenefit);

fprintf(resultFID, '\nAverage Total Benefit after 20 runs: %.2f\n', avgBenefit);

fclose(resultFID); % Save and close the result file

end

function assignmentMatrix = execute\_ga(numServers, numUsers, costMatrix, resourceMatrix, capacityLimits)

populationSize = 100;

generations = 300;

crossRate = 0.8;

mutateRate = 0.02;

candidateSolutions = zeros(populationSize, numServers \* numUsers);

for idx = 1:populationSize

candidateSolutions(idx, :) = adjust\_feasibility(rand(1, numServers \* numUsers), numServers, numUsers);

end

fitnessScores = arrayfun(@(i) compute\_fitness(candidateSolutions(i, :)), 1:populationSize);

for gen = 1:generations

parents = perform\_selection(candidateSolutions, fitnessScores);

offspring = apply\_crossover(parents, crossRate);

offspring = apply\_mutation(offspring, mutateRate);

for i = 1:populationSize

offspring(i, :) = adjust\_feasibility(offspring(i, :), numServers, numUsers);

end

newFitness = arrayfun(@(i) compute\_fitness(offspring(i, :)), 1:populationSize);

allCandidates = [candidateSolutions; offspring];

allFitness = [fitnessScores, newFitness];

[~, idx] = sort(allFitness, 'descend');

candidateSolutions = allCandidates(idx(1:populationSize), :);

fitnessScores = allFitness(idx(1:populationSize));

end

[~, bestIdx] = max(fitnessScores);

bestSolution = candidateSolutions(bestIdx, :);

assignmentMatrix = reshape(bestSolution, [numServers, numUsers]);

function score = compute\_fitness(solution)

sol = reshape(solution, [numServers, numUsers]);

total = sum(sum(costMatrix .\* sol));

usedResource = sum(sol .\* resourceMatrix, 2);

overCapacity = sum(max(0, usedResource - capacityLimits));

assignedPerUser = sum(sol, 1);

assignmentPenalty = sum(abs(assignedPerUser - 1));

penalty = 1e6 \* (overCapacity + assignmentPenalty);

score = total - penalty;

end

end

function selectedParents = perform\_selection(population, fitness)

n = size(population, 1);

selectedParents = zeros(size(population));

for i = 1:n

i1 = randi(n); i2 = randi(n);

if fitness(i1) > fitness(i2)

selectedParents(i, :) = population(i1, :);

else

selectedParents(i, :) = population(i2, :);

end

end

end

function offspring = apply\_crossover(parents, rate)

[n, genes] = size(parents);

offspring = parents;

for i = 1:2:n-1

if rand < rate

point = randi(genes - 1);

offspring(i, point+1:end) = parents(i+1, point+1:end);

offspring(i+1, point+1:end) = parents(i, point+1:end);

end

end

end

function mutated = apply\_mutation(pop, rate)

mutated = pop;

for i = 1:numel(pop)

if rand < rate

mutated(i) = 1 - mutated(i);

end

end

end

function fixedSol = adjust\_feasibility(sol, numServers, numUsers)

reshaped = reshape(sol, [numServers, numUsers]);

for j = 1:numUsers

[~, idx] = max(reshaped(:, j));

reshaped(:, j) = 0;

reshaped(idx, j) = 1;

end

fixedSol = reshape(reshaped, [1, numServers \* numUsers]);

end