Code Part

### Fitness.m

function result = fitness(s)

s1 = bin2dec(s(1:7));

s2 = bin2dec(s(8:14));

result = 10^9-(625-(s1-25)^2)\*(1600-(s2-10)^2)\*sin(s1\*pi/10)\*sin(s2\*pi/10);

### Crossover.m

function children = crossover(parents1, parents2, pCrossover)

for i = 1:size(parents1,1)

% If crossover is to be performed

if (rand() < pCrossover)

crossoverPoint = floor(rand()\*(size(parents1,2)))+1;

for j = 1:size(parents1,2)

if (j <= crossoverPoint)

children(i\*2-1,j) = parents1(i,j);

children(i\*2,j) = parents2(i,j);

else

children(i\*2-1,j) = parents2(i,j);

children(i\*2,j) = parents1(i,j);

end

end

else

children(i\*2-1,:) = parents1(i, :);

children(i\*2,:) = parents2(i, :);

end

end

### Mutation.m

function children = mutation(currChildren, pMutation)

% For each bit

for i = 1:size(currChildren,1)

for j = 1:size(currChildren,2)

% If we satisfy the probability

if (rand()<pMutation)

% Flip the bit.

if (currChildren(i,j) == '0')

currChildren(i,j) = '1';

else

currChildren(i,j) = '0';

end

end

end

end

children = currChildren;

### selection\_R.m

% selects numPairs pairs of parents from population using fitness

% proportional selection

% returns two parent matrix of size numPair x 14

function [parent1, parent2] = selection\_R(population, numPairs)

%size(population, 1)

sizePopulation = size(population, 1);

%cache fitness so we don't have to recalculate

fitnessCache = zeros(sizePopulation, 1);

for i = 1: sizePopulation

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

end

sumFitness = sum(fitnessCache);

parent1(1, 1:14) = ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0'];

parent2(1, 1:14) = ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0'];

% for each pair

i = 1;

while (i <= numPairs)

% start at 0, at each individual, add fitness(individual)/sumFitness

% and see if it's less than the rand number, if it is, choose it

currentNum = 0;

% r is sorted so that you can look for parent 1 first and then parent 2

% for computational efficiency (since order doesn't matter)

r = sort(rand(2,1));

% tells you first parent is picked

firstParentPicked = false;

for j = 1 : sizePopulation

currentNum = currentNum + fitnessCache(j)/sumFitness;

if ~firstParentPicked

if r(1) <= currentNum

parent1(i, 1:14) = population(j, :);

firstParentPicked = true;

end

else

if r(2) <= currentNum

parent2(i, 1:14) = population(j, :);

break;

end

end

end

% different individuals are used for parent 1 and parent 2

if (size(parent1,1) == size(parent2,1))

i = i + 1;

end

end

### selection\_T.m

% selects numPairs pairs of parents from population using fitness

% proportional selection

% returns two parent matrix of size numPair x 14

function [parent1, parent2] = selection\_T(population, numPairs)

sizePopulation = size(population, 1);

%cache fitness so we don't have to recalculate

fitnessCache = zeros(sizePopulation, 1);

for i = 1: sizePopulation

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

end

% for each pair

for i = 1 : numPairs

r = randperm(sizePopulation);

if fitnessCache(r(1)) < fitnessCache(r(2))

parent1(i,:) = population(r(2), :);

else

parent1(i,:) = population(r(1), :);

end

if fitnessCache(r(3)) < fitnessCache(r(4))

parent2(i,:) = population(r(4), :);

else

parent2(i,:) = population(r(3), :);

end

end

### GA.m

### (Note: GA does not change for selection\_R/T, as we passed in another parameter, selectionMethod, to distinguish which selection method to use. Therefore, we did not have to change GA in order to perform different selection method)

function [solution, sbest]= GA(Xinitial, popsize, maxGen, pCrossover, pMutation, selectionMethod)

population = Xinitial;

fitness\_pop = zeros(popsize, 1);

fitness\_children = zeros(popsize,1);

solution = zeros(maxGen,3);

sbest = zeros(maxGen,2);

for iter=1:maxGen

% Calculate fitness for every sample

for i=1:popsize

fitness\_pop(i) = fitness(population(i,:));

end

% Update the average fitness for last iteration

if (iter > 1)

solution(iter-1,2) = mean(fitness\_pop);

end

if (strcmp(selectionMethod, 'selection\_T'))

[parents1, parents2] = selection\_T(population, popsize/2);

elseif (strcmp(selectionMethod, 'selection\_R'))

[parents1, parents2] = selection\_R(population, popsize/2);

else

fprintf('check your inputs');

end

children = crossover(parents1, parents2, pCrossover);

children = mutation(children, pMutation);

% Calculate fitness for each children

for i=1:popsize

fitness\_children(i) = fitness(children(i,:));

end

[maxpVal, maxpIndex] = max(fitness\_pop);

[mincVal, mincIndex] = min(fitness\_children);

[maxcVal, maxcIndex] = max(fitness\_children);

% Put the answer into solution and sbest

solution(iter,1) = iter;

if (maxcVal > maxpVal);

sbestIter = bin2dec(children(maxcIndex,:));

solution(iter,3) = maxcVal;

else

sbestIter = bin2dec(population(maxpIndex,:));

solution(iter,3) = maxpVal;

end

sbest(iter, 1) = floor(sbestIter / 128);

sbest(iter, 2) = mod(sbestIter, 128);

% Pick the best ones to stay

children(mincIndex,:) = population(maxpIndex,:);

population = children;

end

for i=1:popsize

fitness\_pop(i) = fitness(population(i,:));

end

solution(maxGen,2) = mean(fitness\_pop);

### generatePopulation.m

function population = generatePopulation(populationSize)

populationIndex = randi(2, populationSize, 14);

choice = ['0', '1'];

for i = 1:populationSize

for j = 1:14

population(i,j) = choice(populationIndex(i,j));

end

end

### oneb.m(Used to calculate result for part 1.b)

noSimulations = 30;

popsize = 20;

Xinitial(1:popsize,1:14,1) = generatePopulation(popsize);

for i = 2 : noSimulations

Xinitial(1:popsize, 1:14, i) = generatePopulation(popsize);

end

maxGen = 50;

pCrossover = 0.9;

pMutation = 0.05;

[solution, sbest]= GA(Xinitial(:, :, i), popsize, maxGen, pCrossover, pMutation, 'selection\_R');

for i = 2:noSimulations

[solution\_, sbest\_]= GA(Xinitial(:, :, i), popsize, maxGen, pCrossover, pMutation, 'selection\_R');

solution(:,:,i) = solution\_;

sbest(:,:,i) = sbest\_;

end

x = 1: maxGen;

y = mean(solution(:, 3, :), 3);

figure;

plot(x,y, 'r');

xlabel('generation')

ylabel('mean of best fitness')

title('mean over 30 runs of best fitness vs. generation');

hold on

globalMax = 1088359375;

globalMaxCount = 0;

for i = 1:noSimulations

%plot(x,solution(:,3,i), ':');

if solution(maxGen, 3, i) == globalMax;

globalMaxCount = globalMaxCount + 1;

end

end

fprintf('%d of %d runs found the global maximum\n', globalMaxCount, noSimulations);

### twob.m(Used to generate results for part 2.b)

noSimulations = 30;

popsize = 20;

Xinitial(1:popsize,1:14,1) = generatePopulation(popsize);

for i = 2 : noSimulations

Xinitial(1:popsize, 1:14, i) = generatePopulation(popsize);

end

maxGen = 50;

pCrossover = 0.9;

pMutation = 0.05;

[solution1, sbest1]= GA(Xinitial(:, :, i), popsize, maxGen, pCrossover, pMutation, 'selection\_T');

[solution2, sbest2]= GA(Xinitial(:, :, i), popsize, maxGen, pCrossover, pMutation, 'selection\_R');

for i = 2:noSimulations

[solution\_, sbest\_]= GA(Xinitial(:, :, i), popsize, maxGen, pCrossover, pMutation, 'selection\_T');

solution1(:,:,i) = solution\_;

sbest1(:,:,i) = sbest\_;

[solution\_, sbest\_]= GA(Xinitial(:, :, i), popsize, maxGen, pCrossover, pMutation, 'selection\_R');

solution2(:,:,i) = solution\_;

sbest2(:,:,i) = sbest\_;

end

x = 1: maxGen;

y1 = mean(solution1(:, 3, :), 3);

y2 = mean(solution2(:, 3, :), 3);

figure;

hold on

plot(x,y1, 'r');

plot(x,y2, 'b');

xlabel('generation')

ylabel('mean of best fitness')

title('mean over 30 runs of best fitness vs. generation for both selection methods');

legend('selectionT', 'selectionR');

### three.m(Used to generate results for part 3, this combines with the code from HW2)

### Note: This part already changes the output of GA with 20 multiplexer, so that the plot is comparable with the result from SA.

noSimulations = 30;

popsize = 20;

Xinitial(1:popsize,1:14,1) = generatePopulation(popsize);

for i = 2 : noSimulations

Xinitial(1:popsize, 1:14, i) = generatePopulation(popsize);

end

maxGen = 50;

pCrossover = 0.9;

pMutation = 0.05;

[solution1, sbest1]= GA(Xinitial(:, :, i), popsize, maxGen, pCrossover, pMutation, 'selection\_T');

[solution2, sbest2]= GA(Xinitial(:, :, i), popsize, maxGen, pCrossover, pMutation, 'selection\_R');

for i = 2:noSimulations

[solution\_, sbest\_]= GA(Xinitial(:, :, i), popsize, maxGen, pCrossover, pMutation, 'selection\_T');

solution1(:,:,i) = solution\_;

sbest1(:,:,i) = sbest\_;

[solution\_, sbest\_]= GA(Xinitial(:, :, i), popsize, maxGen, pCrossover, pMutation, 'selection\_R');

solution2(:,:,i) = solution\_;

sbest2(:,:,i) = sbest\_;

end

x = 1: popsize: popsize\*maxGen;

y1 = 1./mean(solution1(:, 3, :), 3);

y2 = 1./mean(solution2(:, 3, :), 3);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

sinitial = floor(rand(noSimulations,2)\*128);

solutionset1 = [];

solutionset2 = [];

sbestset1 = [];

sbestset2 = [];

T01 = 231267670;

T02 = 68315651;

alpha1 = 0.9967;

alpha2 = 0.9979;

cputimeset = zeros(noSimulations,1);

for i = 1:noSimulations

cputimeset(i) = cputime;

[solution1\_, sbest1\_] = SA(sinitial(i, :), T01, alpha1, 1, 1, 1000);

cputimeset(i) = cputime - cputimeset(i);

[solution2\_, sbest2\_] = SA(sinitial(i, :), T02, alpha2, 1, 1, 1000);

solutionset1(1:size(solution1\_,1), 1:size(solution1\_,2), i) = solution1\_;

sbestset1(1:size(sbest1\_,1), 1:size(sbest1\_,2), i) = sbest1\_;

solutionset2(1:size(solution2\_,1), 1:size(solution2\_,2), i) = solution2\_;

sbestset2(1:size(sbest2\_,1), 1:size(sbest2\_,2), i) = sbest2\_;

end

avgsolution1 = mean(solutionset1,3);

avgsolution2 = mean(solutionset2,3);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure;

hold on

plot(x,y1, 'r');

plot(x,y2, 'b');

plot(avgsolution1(:,1), avgsolution1(:,3), 'c--');

plot(avgsolution2(:,1), avgsolution2(:,3), 'g--');

xlabel('evaluations')

ylabel('mean of best fitness')

title('mean over 30 runs of minimization vs. evaluations for hw2 and 3');

legend('GA: selectionT', 'GA: selectionR', 'SA: 4b', 'SA: 4c');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

globalMin = 891015625;

globalMinCount = zeros(4,1);

for i = 1:noSimulations

if 1/solution1(maxGen, 3, i) <= globalMin;

globalMinCount(1) = globalMinCount(1) + 1;

end

if 1/solution2(maxGen, 3, i) <= globalMin;

globalMinCount(2) = globalMinCount(2) + 1;

end

if solutionset1(1000, 3, i) <= globalMin;

globalMinCount(3) = globalMinCount(3) + 1;

end

if solutionset2(1000, 3, i) <= globalMin;

globalMinCount(4) = globalMinCount(4) + 1;

end

end

fprintf('%d of %d runs found the global maximum for GA: selectionT\n', globalMinCount(1), noSimulations);

fprintf('%d of %d runs found the global maximum for GA: selectionR\n', globalMinCount(2), noSimulations);

fprintf('%d of %d runs found the global maximum for SA: 4b\n', globalMinCount(3), noSimulations);

fprintf('%d of %d runs found the global maximum for SA: 4c\n', globalMinCount(4), noSimulations);

fprintf('mean over 30 runs results: \n');

fprintf('GA: selectionT is %f\n', y1(maxGen));

fprintf('GA: selectionR is %f\n', y2(maxGen));

fprintf('SA: 4b is %f\n', avgsolution1(1000,3));

fprintf('SA: 4c is %f\n', avgsolution2(1000,3));

fprintf('standard deviation over 30 runs results: \n');

fprintf('GA: selectionT is %f\n', std(1./solution1(maxGen, 3, :), 0, 3));

fprintf('GA: selectionR is %f\n', std(1./solution2(maxGen, 3, :), 0, 3));

fprintf('SA: 4b is %f\n', std(solutionset1(1000, 3, :), 0, 3));

fprintf('SA: 4c is %f\n', std(solutionset2(1000, 3, :), 0, 3));

### fitness\_minimization.m(Used to change the minimization problem into maximization)

function result = fitness(s)

s1 = bin2dec(s(1:7));

s2 = bin2dec(s(8:14));

result = 1/(10^9-(625-(s1-25)^2)\*(1600-(s2-10)^2)\*sin(s1\*pi/10)\*sin(s2\*pi/10));