### Code Part

##### Cost.m

function result = cost(S)

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

##### Neighbor.m

%find new neighbor in (-25,25)

function snew = neighbor(s)

%get lowerbound so that it doesn't go below 0

lowbound = s - 25;

if lowbound < 0

lowbound = 0;

end

%get higherbound so that it doesn't go above 500

highbound = s + 25;

if highbound > 127;

highbound = 127;

end

snew = floor(rand\*(highbound - lowbound + 1)) + lowbound;

##### Neighbor2D.m(Applying Neighbor.m to calculate a neighbor in 2D)

function snew = neighbor2D(CurS)

snew = [neighbor(CurS(1)), neighbor(CurS(2))];

while ((snew(1) == CurS(1)) && (snew(2) == CurS(2)))

snew = [neighbor(CurS(1)), neighbor(CurS(2))];

end

##### SA.m

% Simulated Annealing

function [solution, sbest] = SA(sinitial, Tinitial, alpha, beta, Minitial, maxiter)

M = Minitial;

T = Tinitial;

CurS = sinitial;

BestS = CurS;

CurCost = cost(CurS);

BestCost = CurCost;

solution = zeros(maxiter, 3);

sbest = zeros(maxiter, 2);

Time = 1;

while (Time <= maxiter)

[CurS\_res, CurCost\_res, BestS\_res, BestCost\_res] = Metropolis(CurS, CurCost, BestS, BestCost, T, M);

solution(Time:Time+M-1,1) = Time:Time+M-1;

solution(Time:Time+M-1,2) = CurCost\_res;

solution(Time:Time+M-1,3) = BestCost\_res;

CurS = CurS\_res(M,:);

CurCost = CurCost\_res(M);

BestS = BestS\_res(M,:);

BestCost = BestCost\_res(M);

sbest(Time:Time+M-1,:) = BestS\_res;

Time = Time + M;

T = alpha \* T; % Update T after M iterations

M = beta \* M;

end

% Metropolis

function [CurS\_res, CurCost\_res, BestS\_res, BestCost\_res] = Metropolis(CurS, CurCost, BestS, BestCost, T, M)

i = 1;

CurS\_res = zeros(M,2);

CurCost\_res = zeros(M);

BestS\_res = zeros(M,2);

BestCost\_res = zeros(M);

while (i <= M)

NewS = neighbor2D(CurS);

NewCost = cost(NewS);

DeltaCost = (NewCost - CurCost);

if (DeltaCost < 0)

CurS = NewS;

CurCost = NewCost;

if NewCost < BestCost

BestS = NewS;

BestCost = NewCost;

end

else

if (rand(1) < exp( -1. \* DeltaCost / T))

CurS = NewS;

CurCost = NewCost;

end

end

CurS\_res(i,:) = CurS;

CurCost\_res(i) = CurCost;

BestS\_res(i,:) = BestS;

BestCost\_res(i) = BestCost;

i = i + 1;

end

##### SAParameter.m(Used to calculate delta cost using Method 2)

Sp = floor(rand(1,2).\*128);

AP = 20;

neighborCosts = zeros(AP+1, 1);

for i = 1:AP

neighborCosts(i) = cost(neighbor2D(Sp));

end

neighborCosts(AP+1) = cost(Sp);

So = min(neighborCosts);

constant = ones(AP+1,1) \* So;

deltaCosts = neighborCosts - constant;

sumDeltaCosts = sum(deltaCosts);

fprintf('average delta costs is: %f\n', sumDeltaCosts/AP);

##### Part4.m(Used to generate plots for part 4)

thirty = 30;

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

solutionset1 = [];

solutionset2 = [];

sbestset1 = [];

sbestset2 = [];

T01 = 231267670;

T02 = 68315651;

alpha1 = 0.9967;

alpha2 = 0.9979;

cputimeset = zeros(thirty,1);

for i = 1:thirty

cputimeset(i) = cputime;

[solution1, sbest1] = SA(sinitial(i, :), T01, alpha1, 1, 1, 1100);

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

[solution2, sbest2] = SA(sinitial(i, :), T02, alpha2, 1, 1, 1100);

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);

%plot average of BestCost and CurCost

figure

hold on

plot(avgsolution1(:,1), avgsolution1(:,2), 'b');

plot(avgsolution1(:,1), avgsolution1(:,3), 'g');

title ('CurCost and BestCost vs iteration for 4b');

legend ('CurCost', 'BestCost');

xlabel ('iteration');

ylabel ('Cost');

figure

hold on

plot(avgsolution2(:,1), avgsolution2(:,2), 'b');

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

title ('CurCost and BestCost vs iteration for 4c');

legend ('CurCost', 'BestCost');

xlabel ('iteration');

ylabel ('Cost');

%report std/avg of BestCost after 1000 iters

fprintf('average BestCost over all 30 runs after 1000 iterations is: %f\n', avgsolution1(1000, 3));

fprintf('std BestCost over all 30 runs after 1000 iterations is: %f\n\n', std(solutionset1(1000, 3, :), 0, 3));

fprintf('average BestCost over all 30 runs after 1000 iterations is: %f\n', avgsolution2(1000, 3));

fprintf('std BestCost over all 30 runs after 1000 iterations is: %f\n\n', std(solutionset2(1000, 3, :), 0, 3));

fprintf('average CPU time is: %f\n', mean(cputimeset));

##### plotFunc.m(Used to plot the function graph)

figure

[x, y] = meshgrid(0:127, 0:127);

costmap = zeros(128,128);

for i = 0:127

for j = 0:127

costmap(i+1, j+1) = cost([i j]);

end

end

surf(x, y, costmap);