CEE 5290/COM S 5722/ORIE 5340 Heuristic Methods for Optimization
Homework 2: Simulated Annealing
Assigned: Friday, September 5, 2014
Due: Friday, September 12, 2014 @ noon

IMPORTANT: In the interest of being able to answer everyone's questions on the HW promptly (this is a big class), please post your questions on the Piazza. Please title and state your questions *clearly* and *concisely* so that other members of the class may also benefit from your questions. The TA will try to address the questions within 24 hours of the questions being posted. Please email the TA only about matters that cannot be done on Piazza like homework extensions and other course administration issues.

NOTE: The Simulated Annealing (SA) algorithm and metropolis procedure given on the course package are reprinted below. You will implement these algorithms directly for questions in this HW. Check Piazza regularly for hints or corrections, if any.

Algorithm Simulated_annealing(S_0, T_0, α, β, *M*, *Maxtime*);
% S_0 or *sinitial* is the initial solution
% *BestS* is the best solution
% T_0 or *Tinitial* is the initial temperature
% α or *alpha* is the cooling rate
% β is a constant
% *M* represents the time until the next parameter update
% *Maxtime* is the maximum total time for annealing process
% *Time* refers to the number of cost function evaluations performed

Begin
$T = T_0$;
$CurS = S_0$;
$BestS = CurS$; % *BestS* is the best solution seen so far
$CurCost = Cost(CurS)$;
$BestCost = CurCost$;
$Time = 0$;
Repeat
Call Metropolis(*CurS*, *CurCost*, *BestS*, *BestCost*, *T*, *M*);
$Time = Time + M$;
$T = \alpha T$; % Update T after M iterations
$M = \beta M$;
Until ($Time \geq Maxtime$)
Return(*solution, BestS*);
End *of Simulated Annealing*

```
Algorithm   Metropolis(CurS, CurCost, BestS, BestCost, T, M);
Begin
   Repeat
    NewS = Neighbor(CurS);    % Return neighbor from user-defined function
    NewCost = Cost(NewS);
    ΔCost = (NewCost - CurCost);
    If (ΔCost < 0) Then
      CurS = NewS;
      CurCost = NewCost;
        If NewCost < BestCost Then
          BestS = NewS;
          BestCost = NewCost;
        EndIf
    Else
      If (RANDOM < e^(-ΔCost / T) ) Then
        CurS = NewS;
        CurCost = NewCost;
      EndIf
    EndIf
    M = M - 1;
  Until (M = 0)
End of Metropolis
```

1. **SA Parameter Selection when cost function range = (MaxCost and MinCost) are known:**

   a) *Use Method 1 to estimate AvgΔCost.* If M*inCost* is taken as a lower bound on *Cost* in the search space, and *MaxCost* the upper bound, assume you know *MaxCost* - *MinCost* = 100. Assuming the distribution of costs is uniformly distributed between *MaxCost* and *Mincost*, what is reasonable estimate of T_0 if you want probability of accepting an uphill move on the first iteration *Pinitial* = 0.4?

   b) Write down a general expression for T_0 in terms of *MaxCost, MinCost* and *P*1

   c) Now write a similar expression for *Tfinal*, the final temperature, in terms of *MaxCost*, *MaxCost*, and *P2* (the probability of accepting an uphill move on the **final** iteration).

   d) Suppose you have the following parameters for a simulated annealing algorithm: T_0 = 100, *Maxtime* = 200, *beta* = 1, *M* = 1. What should the value of the cooling parameter *alpha* be if you want the Probability on the 200th simulated annealing iteration to be 0.001? Use *MaxCost* – *MinCost* as in part a)

   e) Calculate *alpha* assuming same parameters as in part (d) except with M=10.

2. **SA Parameter Selection when you have computed *AP* cost values**

Use Method 2 to estimate AvgΔCost. Assume you are running an SA optimization trial and you have picked AP = 6 points *which are* 1,2,3,4,5,6. The values you have are Cost (j) = 40, 60, 50, 65, 75, 45 for j=1,2,3,4,5,6 respectively. Assume all the points 1 to 6 are neighbors of each other. What value would you take for the intial value S_0 for your SA search? Estimate a value of T_0 that would give you *P1* of 0.9 using Method 2 for estimating AverageΔCost. (Assume the constant M=1.)

3. **SA Implementation:**

Implement the simulated annealing algorithm given on pages 1 and 2 (i.e. the version in the Xeroxed text including corrections). Combine both the Metropolis procedure and simulated annealing procedure in one MATLAB function file called SA.m. For RANDOM, use the MATLAB "rand" function. The header of this function will read:

function [solution, BestS] = SA(sinitial, Tinitial, alpha, beta, Minitial, Maxtime)

solution is a matrix with one row per iteration, and has column 1 = iteration number, column 2 = C*urCost*, column 3 = B*estCost.*
BestS is a vector of the best solution decision variable values

SA will be used to minimize the following two-dimensional cost function for all further questions:

F(S) = 10^9-(625-(s1-25)^2) *(1600-(s2-10)^2)*sin[(s1)*pi/10]*sin((s2)*pi/10)
Where S=[s1 s2]
Constraints: s1 and s2 are both integer-valued in the range $0 \leq$ s1,s2 ≤ 127
NOTE: In the neiborhood function, the NewS should not include the currentS.

Write a Matlab function called cost.m that returns the value of the above function. The input argument should be S (a vector).

Define the neighborhood function using a function called neighbor.m. The neighborhood should be randomly perturb *one of the two* decision variables current value between max(*s*-25,0) and min(*s*+25, 127). Note that the neighborhood function should not select s as a neighbor of itself, i.e. neighbor(s) $\neq$ s. If you wish, it may be easier to code this if you select the decision variable to be perturbed within the SA code and then call neighbor.m to make the one-dimensional perturbation. Note that in general, as problems increase in dimension, the definition of the neighborhood can become more complex.

Submit a printout of the code for SA.m, cost.m and neighbor.m. Debug thoroughly as you will reuse the SA code in future homeworks! If you care to return other output variables from SA.m, such as *scurrent* (perhaps for debugging/interest), please output them to *additional* output variables (not *solution* or *BestS*) that you define in your SA code.

. 4. **Running SA:**

a) Let *beta* = 1, *M* = 1, *Maxtime* = 1100, P1 of accepting an uphill move is to be 0.9, and the probability of accepting an uphill move after the 1000th iteration (*P2*) is to be .05. What should T_o, *T2,* and *alpha* be? (T2 is the temperature after 1000 iterations.) Write a script that calculates an estimate of averageΔ*Cost* for an uphill move by Method 2 with AP=20. Call this script SAparameter.m

b) Use the values of T_o and α from 4a) above. Generate 30 sets of random integer numbers sinitial (where $0 \le s1,s2 \le 127$) and call this set Z. Now run 30 trials of SA algorithm each with starting value So = $sinitial_i$, for i=1…,30 and $sinitial_i$ in Z. (Let *Sinitial* be the initial value of S at iteration 0, then start counting iterations for each trial after the SA algorithm is called) You should NOT recalculate the SA parameters for each trial. Submit a plot of the average of *BestCost* & C*urCost* (averaged over all 30 runs) vs. iterations for the SA algorithm, evaluated at G=1000. Compute and report the average and standard deviation (use the MATLAB command 'std') of *BestCost* over all 30 runs after 1000 iterations. Also report the average CPU time it takes to do one SA run (use the MATLAB command "cputime" or "tic; toc").

c) Now repeat steps 4a, 4b, with *P1*=0.7, while keeping *P2*= 0.05, *beta* = 1, *M* = 1, G=1000 ,*Maxtime* = 1100. For the new value of *P1* you will have to compute a new corresponding *To* and *alpha* based on your sampled average ΔCost from part a). (You can use the same AP points computed in part 4a).
Run the SA 30 times for *P1=0.7* and compare the average of *BestCost* after 1100 iterations for each value of *P1*. Which value of *P1* works best? (you should run SA 30 times using the same initial points from set Z of part 4b above)

d) The simulated annealing runs after 1000 iterations have a probability 0.05 of accepting an uphill move, so iterations between 1000 and 1100 are mostly greedy search. Do you see much improvement during these last 100 iterations? (Compare values at G=1000 and Maxtime=1100.)

When you implement SA, let P continue decreasing from 0.05 after the G=1000th iteration , but calculate the parameters for the SA algorithm P2 at G=1000th iteration to be 0.05.

Please remember to submit all requested scripts to Blackboard. Everything including the m-files (graphs, written responses to questions, etc.) must be submitted in hard copy into the homework box located in 220 Hollister.