Adaptive Online Learning for Disk Spin-Down
Katarina Lebaric
CS242, HW2

Abstract
The paper “Adaptive Disk Spin-Down for Mobile Computers” (Helmbold, Long, Sconyers, Sherrod, UCSC) addresses the issue of minimizing energy consumption for portable computing systems. The adaptive algorithms discussed therein serve to extend battery life by “spinning down” devices when they are not needed. The decision is when to “time-out”, i.e. to spin-down the disk.

This HW uses the algorithm on page 8 of the “Adaptive Disk Spin-Down for Mobile Computers” paper. This HW calculates and plots the energy usage by the algorithm, the optimal algorithm and the best expert (best fixed time out) for different learning rates eta, shifting rates alpha, and for different spacings of the experts (time-outs), namely, linear, harmonic and exponential. This HW also plots the weight of the experts over time under the aforementioned variations. The initial learning rate was 4.0 and the initial shifting rate was 0.08, which were varied, and it was found that these variations had negligible impact on the energy usage as a whole. However, the plots in this HW show that in all variations, the algorithm is more effective than the best-fixed-time-out value.

Algorithms
The algorithms used are as follows, from page 8 of the “Adaptive Disk Spin-Down for Mobile Computers” paper:

1. Timeout for the master algorithm:
   \[ \text{TimeOut} = \frac{\text{sum(weightofExpert}_i \times \text{expert}_i\text{Prediction})}{\text{sum(allExpertWeights)}} \]

2. Energy loss for each expert \( x_i \)
   \[ \text{EnergyofExperti} = \text{idle} \text{ if } \text{idle}<\text{timeout(expertiPrediction)} \]
   \[ \text{timeout(expertiPrediction)}+\text{spindownCost} \text{ otherwise} \]
   \[ \text{LossofExperti} = \frac{\text{EnergyofExperti}-\text{optimal}}{\text{spindownCost}}; \]

3. Energy for optimal algorithm
   \[ \text{Optimal} = \text{idle} \text{ if } \text{idle}<=\text{spindownCost} \]
   \[ \text{SpindownCost} \text{ otherwise} \]

4. Reduction of weights
   \[ \text{WeightPrime}=\text{weight} \times e^{(-\text{learningRate} \times \text{LossofExperti})} \]

5. Share the weights
   \[ \text{Pool} = \text{sum(weightPrime}*(1-(1-\text{shiftingRate})^{\text{LossofExperti}}) \]
   \[ \text{WeightNew}=(1-\text{shiftingRate})^{\text{LossofExperti}} \times \text{weightPrime}+1/n*\text{pool} \]
Methods
The algorithms were implemented in Matlab using the idle data given on the class web page (cello1). The source code is attached to the report. In order to reduce the runtime, I scaled down the dataset by a factor of 10. Parameters varied included timeout, learning rate and shifting rate, and the result of the optimal algorithm was shown for all variations as the theoretical lower bound.

Results

Summary
The following results are for variations of the spacing of timeouts. Unless otherwise stated, they use the original values for the learning rate and shifting rate (4.0 and 0.08 respectively). Associated plots include the total energy of the algorithm, and optimal energy (in each plot, the energy of the master algorithm is in red and the optimal is in blue) and energy of the experts (in another plot, due to the different scale – this is sometimes done through a log function), and weight over time (in a third plot). In the weight over time plots, time goes in reverse order on the y axis (earliest time is highest y) and experts are in reverse order on the X axis (small x is larger timeouts, large x is smaller timeouts), and the blue part is the weight (as we go through time, we are weighting more to this type of timeouts). In the best spacing of timeouts (which is found to be the harmonic case – the exponential was found to be the worst spacing), the learning rate and shifting rate are varied (first the learning rate is set to 8, and then, along with this, the shifting rate is set to 0.01). It is concluded that the changes in the shifting rate and learning rate have negligible impact on the energy usage of the algorithm.

Plots
1. Linear spacing of timeouts
   a. Energy of the Master and Optimal
b. Energy of the Experts (minimum is the best expert), using log (in the code, this is `plot(log(expertEnergySum))`)

c. Weight over time
2. Exponential spacing of timeouts
   a. Energy of the Master and Optimal

   b. Energy of the Experts (minimum is the best expert)
c. Weight over time

3. Harmonic spacing of timeouts
   a. Energy of the Master and Optimal
b. Energy of the Experts (minimum is the best expert)

c. Weight over time
4. Harmonic spacing of timeouts, with higher learning rate
   a. Energy of the Master and Optimal
      - Energy of the Optimal Algorithm (blue) and Energy of the ML Alg (red) vs. spin-down Cost
      - Total Energy over Sequence

b. Energy of the Experts (minimum is the best expert)
c. Weight over time

5. Harmonic spacing of timeouts, with higher learning rate and smaller shifting rate
   a. Energy of the Master and Optimal
b. Energy of the Experts (minimum is the best expert)

c. Weight over time
%To load data: on cmd line, say: load idlet
%Learning Parameters
eta = 8; %originally 4, change to 8 for variation
alpha = .01; %can adjust eta and alpha as an exercise (originally 0.08)

% go through all the spindown costs
%for s = 1:10  % s can go up to 20, based on the data
for s = 1:20
  s
  %Spin Down Cost
  spindownCost = 1000000*s;

%Initialize weights and time out experts (all are 1/n, i.e. 1/100)
w = ones(1,100)/100;

%Uniform (linear) spacing of experts (these are the k's, i.e., each possible timeout)
x = 0:spindownCost/99:spindownCost; %this means from 0 to spindownCost, in increments of spindownCost/99

%Harmonic spacing of experts
x = spindownCost*ones(1,100)./[1:100];

%Exponential spacing of experts
%x = spindownCost*ones(1,100)./(2.^[1:100]);

%Compute Time Out
TO=(w*x')/sum(w);

Opt = min(spindownCost,idlet); %Opt as a function of all idle times
ExpertEnergySum = zeros(1,100); % Accumulator for idletimes
MLEnergySum(s) = 0; %energy of the algorithm as a function of s (i.e., over all spindownCost)

n=0; %counter variable used to help in plotting weights over time (below)

%NUM IDLETIMES
%for t = 1:round(length(idlet)/80)  %loop over idle times (can scale for speed, or remove the length and 80 stuff and loop over ALL idle times)
for t = 1:round(length(idlet)/10)
    Energy = ones(1,100)*idlet(t); %set energy to idletime as a default
    %as per step 2 on page 8 of the paper, but at the expert level. Uses matrix substitution:
f = find(x < idlet(t));  %find indices of those where timeout is less than idletime (the else case in step 2)
    Energy(f) = x(f)+spindownCost;       %and add spindownCost to them (since by definition the experts are timeouts)
    ExpertEnergySum = ExpertEnergySum+Energy; %accumulate energy for this idle time onto the accumulator

    %as per step 2 on page 8 of the paper, but at the algorithm level, accumulating data for each spindownCost for the expert
    if idlet(t) > TO
        MLEnergySum(s) = MLEnergySum(s) + TO + spindownCost;
    else
        MLEnergySum(s) = MLEnergySum(s) + idlet(t);
    end

%end of step 2 in paper (page 8)
Loss = (Energy-Opt(t))/spindownCost;

%step 3 in paper (page 8):
wprime = w.*exp(-eta*Loss);

%step 4 in paper, page 8:
pool = sum(wprime.*(1-(1-alpha).^Loss));
w = wprime.*(1-alpha).^Loss + 1/100*pool;
%w = wpp./(sum(wpp)); %normalization, not needed

%For Plotting Weights over time. To plot this, do imagesc(WT) at the cmd line
if mod(t,10)==0  %this means to plot for every 10th idletime
    n=n+1;
    WT(n,:) = w;
end

%now update timeout (since the weights changed) (implied in the paper)
TO=(w*x')/sum(w);
end

%NUM IDLETIMES
%OptSum(s)=sum(Opt(1:round(length(idlet)/80))); %optimal alg's energy usage for all
idle times, should be just sum(Opt), no need for scaling;
%OptSum(s)=sum(Opt);
OptSum(s)=sum(Opt(1:round(length(idlet)/10)));
end

%total energy of the algorithm
%UNCOMMENT:
plot(MLEnergySum,'r');
hold on;
plot(OptSum,'b');
title('Energy of the Optimal Algorithm (blue) and Energy of the ML Alg (red) vs.
spindownCost');
xlabel('Expert Index i');
ylabel('Total Energy over Sequence');

%total energy of the experts (if you were to just take 1 expert, not do weighting. Can find
the best expert as the lowest point on the plot):
%UNCOMMENT:
%plot(log(ExpertEnergySum));
%title('Energy of the Experts for all idletimes (lowest point is best expert)');
%xlabel('Expert Index i');
%ylabel('Total Energy over Sequence');

%TO DO: plot total weight over time:To plot this, do imagesc(WT) at the cmd line
%a fancier plot: surf(WT)
%plot explanation: time goes in reverse order on the y axis (earliest time is highest y)and
experts are in reverse order on the X axis (small x is larger timeouts, large x is smaller
timeouts)blue is the weight