CS577 Assignment 3

Liang Wang, Yan Zhai, Lei Kang

March 29, 2013

GREEDY ALGORITHMS

PROBLEM 1

Given a set of n jobs with a processing time t_i and a weight w_i for each job. We want to minimize the weighted sum of the completions times, $\sum_{i=1}^{n} w_i C_i$. We do the followings,

we get the value of weight over time by $\frac{w_i}{t_i}$, where $1 \leq i \leq n$. To select one job each time, we select job j with the largest $\frac{w_j}{t_j}$, which means for all i in the job list, we have $\frac{w_j}{t_j} \geq \frac{w_i}{t_i}$, and recursively do this until all the jobs are done. This simple algorithm can lead to an optimized solution to this problem.

Proof. We prove the correctness of our algorithm by exchange argument. To do this, let's suppose that there is an optimal solution O that differs from our solution S. In other words, S consists of the weight to time fraction $\frac{w_i}{t_i}$ sorted in non-increasing order. So this optimal solution O must contain an inversion-that is, there must exist two neighbouring jobs i and j such that the $\frac{w_i}{t_i} < \frac{w_j}{t_j}$.

We claim that by exchanging these two purchases, we can strictly improve our optimal solution, which contradicts the assumption that O was optimal.

Let us assume the previous time cost until job i is t_x and the weighted sum is N_x. So, the weighted sum of the optimal solution is $N_O = N_x + (t_i + t_x) * w_i + (t_i + t_j + t_x) * w_j$, and that of the exchanged algorithm is $N_S = N_x + (t_j + t_x) * w_j + (t_i + t_j + t_x) * w_i$.

To compare if the exchanged version is better or not, we have $N_S - N_O = t_j * w_i - t_i * w_j < 0$, which shows that the exchanged version has a smaller weighted sum and so is a better solution.

This concludes the proof of correctness. The running time of the algorithm is $O(nlogn)$, since the sorting takes that much time and the rest (outputting) is leaner. So the overall running time is $O(nlogn)$ □

PROBLEM 2

Given a connected graph $G = (V, E)$. Each each e has a time varying edge cost given by function $f_e(t) = a_e * t^2 + b_e * t + c_e$, where $a_e > 0$. We want to get the minimum spanning tree at unknown time t. Let m and n represents the number of edges and the number of nodes, respectively.

Suppose we get a spanning tree with $n-1$ edges, then the sum of the weights is $\sum_{i=1}^{n-1}(a_i t^2 + b_i t + c_i)$ The minimum value of this sum when $t = -2 * \frac{\sum_{i=1}^{n-1} b}{\sum_{i=1}^{n-1} a}$ if $\sum_{i=1}^{n-1} b < 0$ or $t = 0$ if $\sum_{i=1}^{n-1} b \geq 0$.

Now we discuss how to get the $n-1$ edges which has the minimum cost among all the edges. For a given time point, e.g., t_0, we can easily get the top $n-1$ edges by applying Prim's Algorithm. The real question is when any one of the edges among the top $n-1$ edges will be exchanged with any other edges during the change of time t. It only happens when the weight of one edge in the top $n-1$ edges is larger than that of any one edge among the rest. In other words, there is an intersection on these two curves of changing weight. There are totally $\frac{m(m-1)}{2}$ intersections, for each interval among these cross points, we calculate the minimum spanning tree correspondingly, and find the overall minimum weight sum among $\frac{m(m-1)}{2}$ intervals.

PROBLEM 3

Given $\rho \in [0,1]$, we want to find a set $I \subseteq \{1,...,n\}$ of size $n-k$ such that $\frac{\sum_{i\in I} s_i}{\sum_{i\in I} m_i} \geq \rho$, or we want to ensure the selected set after dropping is fulfil $\sum_{i\in I}(s_i - \rho * m_i) \geq 0$.

For each assignment score, with the changes of ρ values, we can get a curve by $y = s_i - \rho * m_i$. There will be $O(n^2)$ number of cross points, and each point is a different ρ value. We get all the values into a vector P. We sort the vector in $O(n^2 logn)$ time. For each value in vector P (or we can do binary search in vector P in $O(logn)$), we sort all the assignment scores in ascending order according to the value of $s_i - \rho * m_i$, if the top $n-k$ values is not smaller than 0, then we get the final $n-k$ assignment scores.

Proof. The essential thing here is that the vector P contains all possible ρ values we need. Note that we only need the top $n-k$ values, the top $n-k$ assignments will be changed only when one curve represented by $y = s_i - \rho * m_i$ in the selected assignment result set goes below another curve that has not been selected yet. Therefore, the vector P contains all the intervals that the top $n-k$ values would be possibly swapped out by any of the rest k values.

For a given ρ, we can find if it meets our requirements by $O(nlogn)$. There are n^2 possible ρ values, and we do binary search. it is $n(log^2 n)$. To find all possible ρ, we find all intersection points by $y = s - m * \rho$, so there are $O(n^2)$ points, we sort it in $O(n^2 logn)$. So, the time complexity is $O(n^2 logn)$.

□

DYNAMIC PROGRAMMING

PROBLEM 1

Let r_i denotes the number of leftover trucks in month i, where $0 \leq r_i \leq S$ and $0 \leq i \leq n$, and $OPT(i, r_i)$ denotes the value of the optimal solution on month i when there are r_i leftover trucks. So $OPT(0, r_i) = 0$ a the start, and the problem we want to address is $OPT(n, 0)$.

In the first month, we need to order trucks anyway, the number of ordered trucks can be calculated by $d_i + r_i$, and the cost is $K + C * r_i$.

To get the solution, the sub problem we want to solve is $OPT(i, r_i) = OPT(i-1, r_{i-1}) + x * K + r_i * C$, where $x = 0$ if $r_i + d_i - r_{i-1} = 0$ and $x = 1$ if $r_i + d_i - r_{i-1} > 0$. By iteratively solving the sub problems over each month, we can get the smallest $OPT(n, r_n)$ as the optimal solution. We need to track all possible leftover trucks each month, so the time complexity is $O(nS)$.

PROBLEM 2