COMS E6232 - Problem Set #2

Alex Wong (asw2181) - asw2181@columbia.edu

March 3, 2017

Problem 1

a. Consider an instance where we have two items x and y where $s_x = \epsilon$, $v_x = 2\epsilon$, $s_y = B$, $v_y = B$, and $\epsilon << 1$. The v_i/s_i ratio of item x is 2 and item y is 1. Thus, the Greedy algorithm will always pick item x regardless of how large B is and regardless how small ϵ is. So as B gets larger and/or ϵ gets smaller, the approximation ratio of the algorithm will increase, thus showing that the approximation ratio of Greedy is not bounded by any constant. □

b. **Theorem 1b** The Modified Greedy algorithm achieves approximation ratio 2.

Proof: Let OPT be the optimal solution that has maximum value where $v(OPT) = \sum_{i \epsilon OPT} v_i$ subject to $\sum_{i \epsilon OPT} s_i \leq B$. We first assume that the items are ordered in a non-increasing fashion according to the ratio v_i/s_i. Let's call the first item that does not fit in the knapsack using the Greedy algorithm as item m. We know that item m's v_i/s_i ratio $\geq$ items $m+1, ..., n$'s v_i/s_i ratio. Thus, if we are able to fit some fraction of item m so that it fills up to the capacity of the knapsack, that solution will be $\geq$ OPT. We can define the fraction of item m that fits into the knapsack as α where $\alpha = (B - \sum_{i=1}^{m-1} v_i)/s_m$. Thus, $OPT \leq (\sum_{i=1}^{m-1} v_i) + \alpha v_m$. Since α is some fraction ≤ 1, we can also say that:

$$OPT \leq (\sum_{i=1}^{m-1} v_i) + \alpha v_m \leq (\sum_{i=1}^{m-1} v_i) + v_m$$

From the inequality above, $\sum_{i=1}^{m-1} v_i$ or v_m must be at least $OPT/2$, showing that the Modified Greedy algorithm will always get a solution at least $OPT/2$, thus achieving an approximation ratio 2. □

Problem 2

a. The lower bound for OPT is $max(max_i(p_i), \frac{\sum_{i=1}^{n} p_i}{m})$.

For $m = 2$ machines and 5 jobs with processing times 3,3,2,2,2, the LPT algorithm will schedule 3,2,2 on m_1 and 3,2 on m_2, giving a makespan of 7. For OPT, we know that a lower bound is max $= max(3, 12/2) = 6$. We can achieve OPT by scheduling 2,2,2 on one machine and 3,3 on the other machine.

For $m = 3$ machines and 7 jobs with processing times 5,5,4,4,3,3,3, the LPT algorithm will schedule 5,3,3 on m_1, 5,3 on m_2, and 4,4 on m_3, giving a makespan of 11. For OPT, the lower bound is $max(5, 27/3) = 9$. We can achieve OPT by scheduling 3,3,3 on one machine and 5,4 on each of the other two machines.

b. If $p_n > OPT/3$, then $3p_n > OPT$, showing that no machine can process more than 2 jobs or else it would be > OPT, which would contradict OPT being the optimal solution. From this, we know that $n \leq 2m$, thus the largest m jobs will first get scheduled on each of the m machines and the rest of the $n - m$ jobs will be assigned to the machine that has the least load at the point of assignment, thus showing that the LPT schedule is optimal. □

c. **Theorem 2c** LPT achieves an approximation ratio of 4/3.

Proof: Let's define job j as the job that finishes last in the LPT schedule and t_j as the span of time from time 0 until the time that job j starts. We know that in the timespan of t_j, $m \cdot t_j$ amount of processing has been done. This amount of processing can't be more than the total amount of processing of all jobs, thus $m \cdot t_j \leq \sum_{i=1}^{n} p_i \implies t_j \leq \frac{\sum_{i=1}^{n} p_i}{m}$. As we saw earlier in part a, $\frac{\sum_{i=1}^{n} p_i}{m}$ is essentially a lower bound for OPT, thus we can say that $t_j \leq OPT$. Then by adding p_j to t_j, we get the makespan of the machine that has scheduled job j, and we can say that $t_j + p_j \leq OPT + p_j$. What the inequality tells us is that the processing time of job j indicates how well the LPT algorithm performs. Referring to part b, if $p_j > OPT/3$, the LPT schedule is optimal. But in the worst case, if $p_j = OPT/3$, then LPT will give an $OPT + p_j = OPT + OPT/3 = 4/3OPT$ makespan, thus showing that LPT achieves an approximation ratio of 4/3. □

d. **Theorem 2d** The ratio of 4/3 of LPT is asymptotically tight as $m \to \infty$.

Proof: We generalize the examples of part a: given m machines, we have 3 jobs that have a processing time of m, and 2 jobs for each processing time $2m-1, ..., m+1$ (if $2m-1 = m+1$, then there are only 2 jobs for both $2m-1$ and $m+1$) giving us a total

of 2m+1 jobs. With LPT scheduling, we find that every machine will get scheduled two jobs with a total processing time of $3m-1$ with exception of the first machine that will get scheduled with 3 jobs with a total processing time of $3m-1+m = 4m-1$. Thus, the makespan with LPT scheduling is $4m-1$. For an optimal makespan OPT, we first schedule all 3 jobs with processing time m on the first machine, and then use LPT scheduling for the rest of the jobs. This will give each machine a total processing time of $3m$, thus making the makespan of $OPT = 3m$. Without loss of generality, the figure below illustrates the makespan of LPT and OPT of the example from part a when $m = 3$:

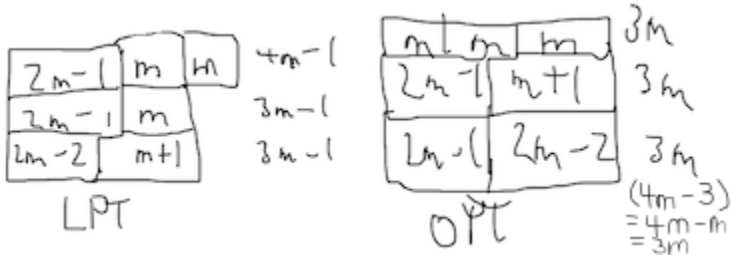

Figure 1: Makespan of LPT = 4m - 1, Makespan of OPT = 3m

Thus, we see the ratio of $LPT/OPT = (4m-1)/3m = \frac{4m}{3m} - \frac{1}{3m} = \frac{4}{3} - \frac{1}{3m}$, showing that the approximation ratio 4/3 is asymptotically tight as $m \to \infty$. □

Problem 3

a. **Proof:** An optimal schedule will have all on-time jobs complete before all late jobs because the objective is to schedule jobs to maximize the total weight of the jobs that complete by their due date; any late jobs' weight will not count towards the total weight, essentially having a weight of 0 in the optimal schedule since they are late. We can also equally reframe the objective of maximizing weight of on-time jobs to minimizing weight of late jobs. By ordering jobs in an earliest due date order, we can achieve minimizing weight of late jobs, which means maximizing the total weight of on-time jobs, thus showing that an optimal schedule has all on-time jobs complete before late jobs and on-time jobs complete in an earliest due date order. □

b. There are n jobs where job j has processing time p_j, a weight w_j, and a due date d_j, where $j = 1, ..., n$. Let $W = \sum_j w_j$ which signifies the maximum possible value an optimal schedule can achieve, where every job completes on time. We use the following Dynamic Programming algorithm to compute the optimal schedule with a maximum total weight of jobs that complete by their due date:

DP Scheduling Algorithm

Preprocessing: We first order the jobs by nondecreasing due date order, which takes $O(nlogn)$ time, where n is the number of jobs to be scheduled. It will not affect the overall runtime since the overall runtime of the algorithm is $O(nW)$ where W has a lower bound of n since each job has weight is a positive integer, making $O(nW) \geq O(n^2)$, which is $> O(nlogn)$.

DP Initialization: Let A be a table that stores values of minimum completion times, $A(w, j)$ represent the minimum completion time of a subset of jobs $1, ..., j$ that are on-time and adding all on-time jobs' weights gives the total weight of w. If there is no feasible subset, then $A(w, j) = \infty$. We initalize $A(0, j) = 0$, for all $j = 0, ..., n$ because to get a total weight of 0, we do not need to have any on-time jobs, thus having a minimum completion time of 0. We initialize $A(w, 0) = \infty$, for all $w = 1, ..., W$ where $W = \sum_j w_j$ because it is not feasible to get some total weight > 0 without having a subset of jobs to schedule.

DP Recursion:
For $1 \leq w \leq W$, $1 \leq j \leq n$, if job j is not an on-time job, we know then that the subset of jobs $1, ..., j-1$ offers the minimum completion time at that weight w. If job j is an on-time job, as in $A(w - w_j, j-1) + p_j \leq d_j$, then we take the minimum completion time between the minimum completion time of subset of jobs $1, ..., j-1$ at weight w or the minimum completion time of subset of jobs $1, ..., j-1$ at weight $w - w_j$ since jobs $1, ..., j-1$ will have at least $w - w_j$ weight, added with the processing time

p_j of job j. The following equation shows the recursion:

For $1 \leq w \leq W, \quad 1 \leq j \leq n$:

$$A(w,j) = \begin{cases} min(A(w,j-1), A(w-w_j, j-1) + p_j) & \text{if } w \geq w_j \text{ and } A(w-w_j, j-1) + p_j \leq d_j \\ A(w,j-1) & \text{otherwise} \end{cases} \tag{1}$$

DP Answer:
Let OPT be the maximum total weight of jobs that complete by their due date. After filling out table A, the maximum weight of on-time jobs is:

$$OPT = max\{w \mid A(w,n) \neq \infty\}$$

which takes $O(n)$ running time. Thus, the total running time is

$$O(nlogn) + O(nW) + O(n) = O(nW)$$

□