Approximation Algorithm

Prof. Palash Dey

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Week – 05

Lecture 25

Lecture 25 : 3 Factor Approximation Algorithm for Scheduling Weighted Jobs on a Single Machine

So, in the last class we have seen a two factor approximation algorithm for scheduling jobs in a single machine for minimizing the sum of completion times. We will see a more general problem where the jobs can have weights and the goal is to minimize the weighted completion time sum of weighted completion times ok. So, that we will study today. rounding based algorithm for minimizing sum of weighted completion times. So, what is the goal? The goal is minimize $\sum w_j C_j$. Now, for the unweighted case we have computed the optimal preemptive schedule and we use that ordering of the of the jobs to schedule the jobs non preemptively.

The problem with weighted setting is that computing the optimal preemptive schedule which minimizes the weighted sum of completion times is NP complete. So, we cannot directly use that approach. So, computing ah schedule which minimizes sum of weighted completion times. is NP complete even when preemption is allowed.

Nevertheless we will see that the insights that we can draw from the proof of two factor approximation algorithm is useful. And this is very typical in algorithm design that often the proof of correctness reveals interesting insights which we can use for designing algorithms. and which is very common which we will see here also. So, first what we will do we will write a linear programming relaxation. So, we write a linear programming relaxation why I am calling it relaxation? Relaxation is a umbrella term to relax or forget about some constraints.

So, the last algorithm for in the algorithm in the last class for the unweighted completion time sum of completion times minimizing that is also relaxation because we let the schedule being preemptive that is that requirement go and then we compute the optimal schedule and then we use that as a guide to design our non-preemptive schedule. We will do the same thing here. So, let us not worry about non-preemptiveness, let us first write down the constraints. and then once we get a optimal solution that solution may be

preemptive also. So, let us see then that we again we will use to guide our non preemptive theorem.

So, here we have variables C_j for each job j denoting its completion time. So, what do we want? The objective function is simple, minimize $\sum w_j C_j$ and what are the constraints we have? So, obviously, any job j cannot be started before its release time and it needs p_j amount of processing time. So, subject to for each job $j \in [n], C_j$ the completion time cannot be less than $r_j + p_j$ and we will have some other interesting constraints. So, we will come back and write here let us first understand. So, you know in order to introduce the second set of constraints we consider any subset S of jobs ok and look at this term $\sum p_j C_j$.

Now, when does this sum is minimized? The sum is minimized only if all the jobs in S is scheduled before any other job outside S. So, this is minimized when all the jobs in S is scheduled before any job in $[n] \setminus S$ ok. Now, look at this thing. So, if that is the case if all the jobs are scheduled before then look at this term $p_j C_j$. Now, C_j is the time when the when the j-th job finishes and what is the earliest that these jobs can start these the earliest these jobs earliest that these jobs earliest time when these jobs can starts is obviously, 0 ok.

Now, these jobs are scheduled in some order. So, let again rename the jobs in S take any optimal schedule optimal non-preemptive schedule of S and rename the jobs. So, that job 1 is scheduled first then job 2 and so on. So, take any optimal schedule for s and rename the jobs, rename the jobs so that $C_1 \leq C_2$ and so on. So, now, you see what is C_1 equal to then will be $p_1 C_1$ is greater than equal to p_1 because it can start later than 0 also.

So, even if it starts at time 0 $C_1 = p_1$ how large yeah what how small C_2 can be $C_2$is greater than equal to $C_1 + p_1$ so which is greater than equal to sorry $p_1 C_1 + p_2$ which is greater than equal to $p_1 + p_2$ and so on. So, if you look at this term $\sum p_j C_j$ Now, p_j gets multiplied with the all p_k's where k-th job is finished before j-th job. I repeat in this term p_j times C_j I am multiplying p_j with all p_k's where the k-th job finishes before the j-th job. Now, before between any two pair of jobs one of them will finish before the other. So, this term is oblivious to the optimal order that is very important.

I do not need to know in which order an optimal schedule for S schedule them. So, this term is can be written as $\sum p_j p_k$. Now, this again can be rewritten as $\frac{1}{2}\left(\sum p_j\right)^2 + \frac{1}{2}\sum p_j^2$ ok. And this is greater than equal to you see $j \leq k$. So, all for all jobs p_j^2 is there.

So, this is distributed here in this $\frac{1}{2}$ and here. So, this is greater than equal to if I remove forget the second part second term this is $\frac{1}{2}$ and this one I am defining it as $p(S)$. define $p(S)$ for any subset of job to be equal to $\sum p_j$. So, $\frac{1}{2}p(S)^2$. So, for every subset of jobs I have this inequality let us write down this inequality now for all subset of jobs $\sum p_j C_j \geq \frac{1}{2}p(S)^2$.

So, this finishes the description of linear programming rounding. Why rounding? Again we are not taking into account various things for example, this any anything in for example, preemption for example, here we are not considering the fact that this schedule for every subset it cannot it cannot schedule all the jobs in S in the best possible way. So, this is a linear programming relaxation of course, an optimal non-preemptive schedules schedule satisfies this, but any schedule any C_js may not correspond to a optimal non-preemptive schedule. Now, what we do we have that LP we solve the LP, let $\{C_j^*\}_{j\in[n]}$ be an optimal solution. Now, because any optimal solution any preemptive schedules satisfies these constraints as we have argued. So, what we have is $\sum w_j C_j^* \leq opt$ ok.

So, again we rename the jobs such that $C_1^* \leq C_2^*$. We assume without loss of generality that the processing time of every job is strictly greater than C_n^* ok. So, again what is the algorithm? The algorithm is we schedule the jobs in this order non preemptively ok. So, we schedule job 1, then job and so on ok. In particular we schedule job 1 from r_1 to $r_1 + p_1$.

We schedule job at $r_1 + p_1$ the machine is free. So, $max\{r_1 + p_1, r_2\}$ from this time to we let it run for p_2 amount of time $max\{r_1 + p_1, r_2\}$. $max\{r_1 + p_1, r_2\} + p_2$ and so on ok. So, next we show that this is a three factor approximation algorithm. So, theorem our schedule has an approximation ratio of at most 3 ok.

So, proof slightly more involved than the earlier one, but not difficult, but before that let us make a comment. Here we see that the number of constraints is exponentially. So, we have exponentially many constraints because we have a we have a constraint for each subset of n and there are 2^n such subsets. if we remove the empty set then $2^n - 1$. So, how do we write it? Even writing it takes exponential amount of time. So, there is a method called ellipsoid method for solving this kind of linear programs in polynomial time.

So, we will see that we will discuss this topic in the next lecture how using ellipsoid

method we can tackle this sort of linear this sort of linear programs which has exponentially many constraints, but polynomially many variables that is very important and this linear programs have some special structure using which we can design what is called a polynomial time separation oracle. So, we will discuss all these things, but for this lecture let us assume that we can compute an optimal solution in polynomial time because that is needed here in our algorithm here ok. So, now let us prove it. So, let as usual let C_j^N be the completion time of job n in our non-preemptive schedule. So, what we will show? We will show that C_j^N is less than equal to $3 \times C_j^*$ for all $j \in [n]$.

This will prove the theorem. this will prove the theorem. Since, ALG is $\sum w_j C_j^N$ this is less than equal to $3 \sum w_j C_j^*$ and that is less than equal to opt this is less than equal to $3 \times opt$. So, let us so, all we need to show is C_j^N is less than equal to $3 \times C_j^*$.

So, here again we observe that like in the proof of non unweighted version of the problem, we observe that the machine is not ideal. in the time interval. Now, again focus on only on the jth job. So, the latest that the j-th job can start is this $\left[max_{\square}^{jr}, C_j^N\right]$ ok. And in this time interval only the jobs in 1 to j can execute and only jobs 1 to can execute in this time interval. So, C_j^N is less than equal to $max_{k=1}^{j} r_k + \sum p_j$ ok. So, this again this term we denote it by $p([j])$. So, this is $max_{k=1}^{j} r_k + p([j])$, this set we call this $[j]$ is this set $\{1, \ldots, j\}$ ok. So, what we will show? So, what we have argued is what we will show that this is less than equal to $3 \times C_j^*$.

So, first look at this term. So, let l be the index which minimizes this. So, such that r_l equal to which maximize this r_l equal to $max_{k=1}^{j} r_k$ ok. So, first observe that C_j^* is greater than equal to C_l^*. that is just by renaming we have renamed the jobs in such a way that $C_1^* \geq C_2^* \geq C_3^* \geq \ldots$ and so on. So, this we have and C_l^* is greater than equal to r_l because we have a constraint because of these constraints.

So, $C_j \geq r_j + p_j$ in particular $C_j \geq r_j$. So, what we have is this that C_j^* is greater than equal to r_l and hence what we have here this is less than equal to C_j^* here see r_l is less than equal to C_j^* and this term remains $p([j])$. This will now next we will show that p this set 1 to j this is at most $2C_j^*$ which will finish the proof. So, to show that $p([j])$ is less than equal to $2C_j^*$ which will prove. So, for that we will use the second set of constraints.

So, consider $S = \{1, \ldots, j\}$ and C^* is a feasible solution and hence it satisfies those

constraints. So, we have the constraints for this set is $\sum_{k \in S} p_k C_k^* \geq \frac{1}{2} p(S)^2$ ok. Now, again by our relabeling we know that $C_j^* \geq C_{j-1}^* \geq \ldots C_1^*$ is how we name the jobs.

$$\frac{1}{2} p([j])^2 \leq \sum_{k \in [j]} p_k C_k^* \leq C_j^* \sum_{k \in [j]} p_k = C_j^* p([j])$$

Now, on the other hand we have this is greater than equal to this because of this LP constraints this is greater than equal to $\frac{1}{2} p([j])^2$. So, again using these two what we have is $p([j])$ is less than equal to $2C_j^*$ which is exactly what I need to show here. Hence what we have here is in the next step less than equal to $C_j^* + 2C_j^*$ which is $3C_j^*$ which shows this and in particular that our algorithm is a three factor approximation algorithm ok. So, let us stop here. Thank you.