Approximation Algorithm

Prof. Palash Dey

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Week – 05

Lecture 24

Lecture 24 : 2 Factor Approximation Algorithm for Scheduling Unweighted Jobs on a Single Machine

welcome. So, far we have seen couple of algorithm design techniques like greedy algorithm, local search, dynamic programming for designing approximation algorithms. So, next we start another very powerful method which is called linear programming rounding. So, this is the next topic. linear programming rounding. As usual let us understand this technique using a using couple of examples.

So, our first example is scheduling jobs on a single machine. So, what is the input? n jobs with release dates $r_1, \ldots, r_n$ let us call it release time. and processing times $p_1, \ldots, p_n$, each job should be started only after its release time machine can run. at most one job at any point of time and the schedule is non preemptive.

that is a job once started should be allowed to finish. before allocating the machine to some other job. So, this kind of schedules are called non preemptive schedule. And what is the goal? What we want to minimize? Let C_j be the completion time. for the j-th job in a schedule.

The goal is to compute a schedule which minimizes summation completion times. Note that this goal is equivalent to minimizing average completion times because average completion time is $\frac{\sum_{j=1}^{n} C_j}{n}$. note that this goal is equivalent to minimizing average completion time. So, there is a weighted version of it where each job has a weight w j and the goal is to minimize the weighted sum of completion time $\sum_{j=1}^{n} w_j C_j$. So, this is a generalization of this unweighted version and what we will see that weighted version needs a linear programming rounding.

But, unweighted version there is a nice algorithm for the unweighted version that not only gives a two factor approximation algorithm, but it gives a guideline and idea of how

to design constant factor approximation algorithm for the weighted version. So, let us first see the unweighted version that means, we want to minimize $\sum_{j=1}^{n} C_j$ all jobs has same weight may be weight equal to 1 and then in the next lecture we will see the more general weighted version. So, what is this? What is the algorithm? So, we know how to minimize these sum of completion times if we are allowed preemption. We can compute a preemptive schedule what is a preemptive schedule? Here we do not need a job once started to run it till it is till it finishes. So, we can in the middle of its of the execution of a job we can take the machine from the job it is like we can pause it take the machine from the job and allocate it to some other machine and then later we can resume the processing of the earlier job.

So, that is called preemption schedule. So, if preemption is allowed there is a simple algorithm which is called shortest remaining time fast which computes a preemptive schedule which minimizes sum of completion times. We can compute a preemptive schedule which minimizes $\sum_{j=1}^{n} C_j$ the algorithm is called we can compute a preemptive schedule in polynomial time. the algorithm is called the shortest remaining processing time SRPT. So, what is the schedule? We start at time 0 and schedule a job which has the smallest remaining processing times.

among the jobs that are released and has not completed yet. ok and we schedule it until either it is complete or some new job is released and then we iterate, we iterate until we execute all jobs completely ok. Let us understand this using an example, suppose I have 3 jobs with the processing time p_1 the of the first job is say 5 unit, p_2 is 3 unit 1 and p_3 is 4 unit and the release times are say r_1 equal to 0, r_2 equal to 1 and r_3 equal to 2. So, at t equal to 0 at time equal to 0 only first job is available. So, in the time from 0 to 1 job 1 is allotted, then at time 2 at time 1 second job is released and among the jobs there are 2 jobs first job and second job.

First job has 4 units of processing time remaining, second job has 3 units of processing time remaining. So, second job is scheduled there t 1 to time 1 to time 2. at time 2 third job is released again we compute what is the smallest remaining time what is which job has the smallest remaining times it turns out that again it is the second job. So, we keep running the second job. till it finishes which is 4 and at time 4 we have 2 jobs both have same processing time remaining.

So, we can schedule arbitrarily. So, maybe I schedule J_1 first from 4 to 8 and then J_3 from 8 to 12. So, this is the outcome of SRPT algorithm. Now you take it as a homework that SRPT which is a greedy algorithm SRPT computes a preemptive which minimizes sum of completion times among all preemptive and note that every non-preemptive

schedule is also a preemptive schedule. So, among all preemptive or non-preemptive schedules.

it is a greedy algorithm and you can use standard proof techniques for design for proving the optimality of greedy algorithms. There are various proof techniques for example, swap cut and paste and various others and it is a easy proof I will leave it as a homework. So, from this what we get is this observation that for any set of jobs if I compute the preemptive schedule and if cjp for any set of n jobs if the or if a SRPT schedule. SRPT schedule need not be unique because whenever there is a tie it is an it can schedule any jobs. For example, in the in the last example at t equal to 4 both job 1 and job 3 has 4 units of processing time remaining.

So, we can pick any arbitrary schedule. So, this is one SRPT schedule another SRPT schedule is $J_1 J_2 J_2$. then here is J_3 and here is J_1 both are SRPT schedule. So, for any set of n jobs pick any SRPT schedule and if that SRPT schedule completes job j at C_j^p, p stands for preemptive then because SRPT is an optimal preemptive schedule and non-preemptiveness can only increase the summation of sum of completion times because every non-preemptive schedule is also a preemptive schedule what we have is $\sum_{j=1}^{n} C_j^p$ this is less than equal to opt ok. Now what is our algorithm? The algorithm is very simple for the non-preemptive schedule which is the two factor approximation algorithm.

Let us write two factor two approximation algorithm for non-preemptive scheduling. The algorithm is you first compute a preemptive schedule an optimal preemptive schedule may be SRPT. So, compute an SRPT schedule. rename the jobs based on how they ah in which order they finish, rename the jobs such that $C_1^p \leq C_2^p \leq \ldots \leq C_n^p$ and execute these jobs execute these jobs in this order in non preemptive schedule. So, this is the first step second step is output the non-preemptive schedule which executes job 1, then job 2, and so on in particular. So, start or formally. So, first job one should be executed formally start execution of job 1 at r_1 whenever it is released and execute it till $r_1 + p_1$. So, let C_j^N be the completion time of job n in the non-preemptive schedule output by the algorithm. That is what is C_1^N? C_1^N is $r_1 + p_1$ ok.

What is C_2^N? C_2^N is max of C_1^N and r_2. So, machine the machine is blocked till C_1^N and at r_2 it is job 2 is released plus p_2 and so on. So, what is ALG? ALG is $\sum_{j=1}^{n} C_j^N$. So, what we show is lemma that for each job for each job $j \in \{1, \ldots, n\}$ this set I denote by $[n]$ this is $\{1, \ldots, n\}$ C_j^N can be at most $2C_j^P$ ok. So, if I prove this then it shows it is a two factor approximation algorithm in particular.

$\sum_{j=1}^{n} C_j^N \leq 2\sum_{j=1}^{n} C_j^P \leq 2\,opt$. So, when it we just need to prove this that some C_j^N is less than equal to $2C_j^P$ and we show this by showing 2 easy lower bound. First you observe that C_j^P is greater than equal to $max_{k=1}^{j} r_k$. So, this is the latest release time among the jobs from 1 to j because in this set 1 to j, jth job is the last job to start and it can start latest at $max_{k=1}^{j} r_k$. This is obvious and also because the jobs 1 to $j-1$ should be finished first.

So, C_j^P should be greater than equal to $\sum_{k=1}^{j} p_k$ ok. So, these are the bounds on C_j^P. Now, by construction how we design this algorithm? by construction C_j^N is greater than equal to $max_{k=1}^{j} r_k$. ok and what is what is C_j^N of course, you see when does what is the earliest time it can it can start it can start at $max_{k=1}^{j} r_k$. So, C_j^N is less than equal to.

So, here is another observation is that the machine can never be ideal in the time interval from $max_{k=1}^{j} r_k$ to and when the jobs finish j-th job finish ok. And what could be the maximum length of this interval? The in this interval only jobs in the set from 1 to j are executed the maximum length of this interval is only this jobs j jobs are processed. So, the maximum it could be $\sum_{k=1}^{j} p_k$. So, what we have is C_j^N is less than equal to $max_{k=1}^{j} r_k + \sum_{k=1}^{j} p_k$ both of them are less than are greater than equal both of them are less than equal to C_j^P these are the inequalities. So, this is less than equal to $2C_j^P$ which shows which proves the lemma and which in particular shows that these are two factor approximation algorithm.

So, in the next class we will see three factor approximation algorithm using linear programming rounding for the weighted version of this problem. Thank you.