Approximation Algorithm

Prof. Palash Dey

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Week – 04

Lecture 19

Lecture 19 : PTAS for Minimizing Makespan for Scheduling Jobs on Constant Number of Machines

Welcome. So, in the last class we have seen a fully polynomial time approximation scheme also known as FPTAS for the knapsack problem. So, our next problem is scheduling jobs on identical machines parallelly and again we will see a PTAS for this problem and we will also see that there is no FPTAS for this problem unless P = NP. So, scheduling jobs on multiple identical parallel machines. So, recall we have already seen this algorithm the input are n jobs, n jobs with processing $p_1, \dots, p_n$, we have m machines compute a schedule which minimizes its makespan. What is max span? It is the last completion time of any job or in saying the same thing in other ways it is the maximum load of any machine.

And we have seen the least scheduling algorithm for this problem which is a simple greedy algorithm. which achieves a two factor approximation algorithm. And then we have seen a version of least scheduling where we sort the jobs from highest processing time to smallest processing time and again apply the same greedy algorithm and we have seen it is a four third factor approximation algorithm ok. So, now we will slightly modify that algorithm and design a PTAS fully not fully polynomial time approximation scheme for this problem.

For that what we will design is that for we will design for every integer k and algorithm say A_k. which provides a $1+\frac{1}{k}$ factor approximate schedule. you see this is enough to get a PTAS if the runtime is within the PTAS boundary, but approximation factor wise this is good because if I set k to be $\frac{1}{\epsilon}$ I am getting a $1+\epsilon$ factor approximation algorithm. So, what is the idea? The idea is as we have seen in the four-third factor approximation algorithm that it makes sense to process the longest longer jobs first and more efficiently and we can process the shorter jobs later. So, the idea is to partition the jobs into longer and shorter this sort of these are the two kinds of jobs.

idea divide the jobs into longer jobs and shorter jobs ok. So, what do you mean by longer jobs? So, a job is called long job if a job say J is called a long job if p_j is greater than You recall that we had we use two lower bounds for proving the two factor approximation algorithm for the least scheduling algorithm. One lower bound is average load of any machine. We know the average load it is the sum of the processing times by the number of machines and the other lower bound was the maximum processing time of any job. So, we use the first lower bound to define the long jobs which was the sum of processing times by m.

These are the this is the average load of any machine and we divide it by k. So, we say a job is long job if it is processing time is more than $\frac{1}{km} \times \sum_{i=1}^{n} p_i$. Note that there can be at most $k \times m$ long jobs. Other jobs are called short jobs. job is called short if it is not long ok.

So, the idea is you we find out the optimal schedule for long jobs which are in few in numbers as we have argued the number of long jobs could be at most $k \times m$ and then for the short jobs we use the list scheduling algorithm greedy list scheduling algorithm. So, the algorithm compute an optimal schedule for the long jobs and then use least scheduling algorithm to schedule the short jobs. So, what is the runtime for computing the optimal schedule for long jobs? The number of schedules of long jobs is at most. each job has can be scheduled on one of the m machines. And so, the number of possible schedules is at most m^{km} which is the number of long jobs.

See which is exponential in k only if the number of machines m is a constant. And notice that exponential in k is fine because the run time for PTAS it can be exponential in function of epsilon and k is like $\frac{1}{\epsilon}$. So, this run time is fine. Now, how good is this algorithm? So, let C_{max} be the makespan of the schedule.

output by our algorithm ok. So, again like the two factor approximation analysis of local search algorithm let l be the last job to finish. it may not be unique. So, let us write A ok. So, C_{max} is then less than equal to $p_l + \frac{\sum_{j \in [n], j \neq l} p_j}{m}$.

because time from 0 to $C_{max} - p_l$ all machines were busy executing jobs from 1 to n except l. Now, p_l is less than equal to Now, there are two cases whether l is a short job or long job. If l is a long job then then the schedule output by the algorithm is an optimal schedule because we have scheduled the long jobs optimally. So, if l is a long job then

C_{max} equal to opt since we have scheduled long jobs optimally. So, let us assume that l is not a long job l is a short job.

So, let us assume that l is a short job. So, because l is a short job we know that it is execution time should be $\frac{\sum_{i\in[n]} p_i}{mk}$. So, then we have $C_{max} \leq \frac{\sum_{i\in[n]} p_i}{mk} + \frac{\sum_{i\in[n], i\neq l} p_i}{m}$. Now use the fact that $\frac{\sum_{i\in[n], i\neq l} p_i}{m}$ is a lower bound on opt.

So, this is also lower bound on opt this is less than equal to $\frac{opt}{k} + opt$ which is $(1+\frac{1}{k}) \times opt$. setting $k=\frac{1}{\epsilon}$, we obtain a PTAS for the case when the number of jobs number of machines is constant. So, next what we do we will replace the brute force algorithm for computing the optimal schedule for long jobs with a clever dynamic programming algorithm which runs in polynomial time when k is some constant even for any number of machines when number of machines is an input parameter. So, now, we design dynamic programming algorithm to compute Now, here also we see it is not in not needed to compute the optimal solution exactly, because you know we can have an instance where all the jobs are long jobs and because it is an NP complete problem we do not hope to solve the have a dynamic programming algorithm which runs in polynomial in m time, but still compute the optimal solution. but you know it is enough if it computes an approximately optimal solution and algorithm which is say whose output is $1+\frac{1}{k}$ factor approximation algorithm because we are aiming for a PTAS.

So, now, we design a dynamic programming algorithm to compute on $1+\frac{1}{k}$. because you see here in this case. here it is enough that C_{max} if l is the long job then we do not need C_{max} to be equal to opt it is fine even if C_{max} is less than equal to $opt+\frac{1}{k}$. When l is a short job we already have we are not using the fact that the schedule on long jobs is an optimal schedule and still we are able to show that C_{max} is less than equal to $(1+\frac{1}{k}) \times opt$. we are using that the schedule on long jobs is optimal only here to claim that $C_{max}=opt$, but we do not need $C_{max}=opt$ it is enough if we have $C_{max} \leq (1+\frac{1}{k}) \times opt$.

And that is where we will design a dynamic programming based algorithm design dynamic approximately optimal schedule for long jobs. ok. For that it will be convenient to know if we know whether if we know an optimal value of the solution. So, it will be convenient for designing the algorithm it will be convenient to have a target makespan T for the jobs. Our algorithm will output a schedule of make span $(1+\frac{1}{k})\times opt$ times t if there indeed exists a schedule with make span at most T. Let us call this algorithm give it some name B_k, if there does not exist any with make span at most T then our algorithm reports no reports it. So, the high level idea of this algorithm is again breaking the jobs into long and short further. And here is the catch then we will instead of doing brute force we will do a clever dynamic programming algorithm to reduce the search space substantially and the runtime will be in in exponential in k only irrespective of whether m is constant or not. So, it will be exponential in k times poly in m n.

Once we have this algorithm B_k, how do you get this target schedule T, target make span T, this is very important. here again because you know in the problem we do not have any target makespan, but we claim that if we have an algorithm B_k which can do this job with respect to a target makespan T then also it is enough because then we can do a binary search if we know the range where the optimal lies and we know the range. For example, we know so, let opt be the optimal make span, then we know that opt is greater than equal to we have two lower bounds one is average load which is $\frac{\sum_{i=1}^{n} p_i}{m}$ this is the average load this and opt because opt is integral let us assume we have assumed all p_i values of processing times are integral. So, this we can take sill and maximum processing time p_i, i equal to 1 to n both is a lower bound. So, I can use the max this is one and the upper bound is we know we have a schedule which takes time at most summation the average load of any machine plus the maximum load of any machine of any job.

So, that is the first you know least scheduling algorithm for that for this problem which is a two factor approximation algorithm. So, this gives an upper bound. Now, once we have this range and this range is not big then we can do a binary search and use this algorithm to find out the optimal T not optimal T we will find out an $1+\frac{1}{k}$ approximately optimal schedule ok. So, this we will see in the next class ok. So, let us stop here.