Approximation Algorithm

Prof. Palash Dey

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Week – 05

Lecture 21

Lecture 21 : PTAS for Minimizing Makespan for Scheduling Jobs on Parallel Identical Machines contd.

So, in the last class we have we have started doing the PTAS for scheduling jobs into multiple identical machines to minimize the mix span and we have come to a point of designing an algorithm called B_k. which takes a target make span T and the goal is to find out whether the opt is less than T or if opt is greater than equal to T, then it outputs a schedule with processing time at most $\left(1+\frac{1}{k}\right)\times T$ ok. And then we have discussed that it is enough to schedule only long jobs optimally and now in this lecture we will see nice dynamic programming algorithm for doing that job ok. So, this is PTAS for scheduling jobs on multiple identical machines continue. And we were discussing the algorithm BK which either outputs a schedule with make span.

at most $\left(1+\frac{1}{k}\right)\times T$ sorry, where T is the target make span given as input to B_k or it reports that there is no schedule with make span at most T ok. And for that we have defined long jobs whose processing time is greater than equal to $\frac{T}{k}$ and short jobs whose Poisson time is less than $\frac{T}{k}$ and then we have observed that it is enough to schedule the long jobs. with make span at most $\left(1+\frac{1}{k}\right)\times T$ and then we can run the least scheduling algorithm from that schedule to schedule the short jobs obtaining an overall schedule whose make span is at most $\left(1+\frac{1}{k}\right)\times T$. And towards that what we have done we round the processing time of long jobs to integral multiple to the nearest integral multiple of $\frac{T}{k^2}$ sorry nearest multiple of not be integral nearest integral multiple that means,

$p_i' \in \{0, \frac{T}{k}, \frac{2T}{k^2}, \ldots, T\}$. So, these are the possibilities and then we observe that it is enough to compute the optimal schedule for this rounded down instance and that is that schedule the make span of that schedule with respect to the original processing times is at most $\left(1+\frac{1}{k}\right)\times T$.

So, it is enough. to schedule the round down instance round down jobs optimally ok. The first observation is that number of distinct ah processing times could be at most k^2. So, observe that the number of distinct processing times can be at most k^2. The 0 also cannot be there actually the initial first terms for which this value is less than $\frac{T}{k}$ cannot be there because we are considering only long jobs.

strictly more than $\frac{T}{k}$ cannot be there. So, we can ignore this first definitely the first term and hence the number of distinct processing times would be at most k^2 it will be strictly less than k^2 because we are considering only long jobs whose processing times are greater than equal to $\frac{T}{k}$ this is very important. Now, the number of jobs is n. So, and there are few possibilities think of k as constant then there are constantly many possibilities for processing times. Hence, the number of distinct instances with at most k jobs at most n jobs is at most n^{k^2}.

So, think of like this how many jobs are there with processing time $\frac{T}{k^2}$ for each processing time you note down how many jobs are there from that perspective if you count then this is at most n^{k^2} which is a polynomial in n. if k is a constant which is indeed the case recall we will at the end we will substitute k to be something like $\frac{1}{\epsilon}$. So, the number of instances could be at most $n^{\frac{1}{\epsilon^2}}$ if we are we will substitute k to be $\frac{1}{\epsilon}$. So, we can encode. So, we can encode each instance ah by the number of jobs for each processing time.

Let n_i be the number of jobs with processing time $\frac{iT}{k^2}$. And let $opt(n_1,\ldots,n_{k^2})$ be the minimum number of machines needed to process these jobs with make span at most T. So, now look at from a particular machines point of view, how many possible configurations could be there for a particular machine. So, what is the configuration or given a schedule the or if s_i be the number of jobs with processing time $\frac{iT}{k^2}$ scheduled to a machine, then the tuple $(s_1, s_2, \ldots, s_{k^2})$ is called the configuration of that machine. So, that is called configuration of that machine and now we ask if the make span is T and all are long jobs that means, they are they are processing time is greater than equal to $\frac{T}{k}$ each s_i must be less than equal to $k+1$. must be less than equal to k since all the jobs are long for make span to be at most we must have $s_i \in \{0,1,2,\ldots,k\}$.

This is for all i in this set. These are the set of pricing times n_1 to n_{k^2} ok. We must have s_i in this that means, s_i can have at most $k+1$ different values that s_i can take. Hence, the number of different configurations is at most. it is a tuple of k^2 coordinates and each coordinate can take at most $k+1$ values is at most $(k+1)^{k^2}$ ok.

Now, we run to the we come to the dynamic programming recurrence. So, opt is n_1 to n_{k^2} this is we guess the configuration of any 1 machine say machine 1 it is $1+min_{(s_1,\ldots,s_{k^2})\in C}$ denote the set of all configurations that a machine can take minimum of $opt(n_1-s_1,\ldots,n_{k^2}-s_{k^2})$ ok. updating a table entry takes big O of cardinality C can take at most the number of configurations because we have to go over all configurations. It is a dynamic programming algorithm. So, this table entries are already filled.

So, this is $O((k+1)^{k^2})$ and what is the number of cells in the table number of table entries. is less than equal to the number of instances with at most n jobs which is n^{k^2}. Hence, the running time of the dynamic programming algorithm is $O(n^{k^2})$ cells and each cell it takes $(k+1)^{k^2}$ time to update each cell ok. And now, this is the algorithm we compute the optimal schedule. Now, if we we put k to be equal to $\frac{1}{\epsilon}$ to obtain a schedule with make span at most $(1+\epsilon)\times T$ using which we can actually get $(1+\epsilon)\times opt$ in time.

$O(n^{\frac{1}{\epsilon^2}}(1+\frac{1}{\epsilon})^{\frac{1}{\epsilon^2}})$ in this much time. Notice that we have what we have showed how does the proof of correctness of algorithm B_k follow? We have shown that under the assumption that there is an optimal schedule for long jobs with make span less than equal to T or at most T. this algorithm $B_k$outputs a schedule of with make span at most $\left(1+\frac{1}{k}\right)\times T$. Now, nothing is shown if the optimal make span of this long jobs is more than T. So, 2 things can happen algorithm does this sort of things and it gets a schedule.

So, it can check what is the make span. So, if it got hold of a schedule which make span is at most $\left(1+\frac{1}{k}\right)\times T$ in it outputs which is which it can do. Otherwise if after running the algorithm the make span of the schedule that the algorithm obtains is strictly more than $\left(1+\frac{1}{k}\right)\times T$ that implies that the optimal make span for this long jobs is greater than T because we have shown that if the optimal make span for long jobs is less than equal to T then the make span of the schedule output by this algorithm must be $\left(1+\frac{1}{k}\right)\times T$ ok. So, let us this is the PTAS a next natural question is can we have an FPTAS. can we have an algorithm which outputs $1+\epsilon$ factor approximate solution with time polynomial in n and $\frac{1}{\epsilon}$ this is a PTAS not FPTAS.

So, can we have an FPTAS. The answer is no because this problem is what is called strongly NP complete. So, it is known that this scheduling problem. is strongly NP complete. Now, what does strongly NP completeness mean? It means that even if all input numbers are polynomially bounded by the size of the input, then also the problem remains NP complete. Π is called strongly NP complete.

If there exists a polynomial q such that the such that Πremains NP complete even if all input numbers are at most q times n. where n is the size of the input. Now, these problems are called strongly NP complete and for any problem which is strongly NP complete we do not get could not get a FPTAS under very mild assumption. So, let us let us see the proof for this job scheduling problem.

So, suppose not. So, it is a proof by contradiction what is known is that this scheduling problem is strongly NP complete that means, there exist a polynomial Q such that even if all processing times are less than $q(n)$ then also it is NP complete. So, from here it means NP complete even when Π is less than equal to $q(n)$ for all i in n ok. So, this is the processing times of this and what we can show is that if there exist an FPTAS we can use that algorithm to solve this problem to have a polynomial time algorithm for scheduling problem under this assumption that the processing times are less than equal to $q(n)$. So, suppose there exist an algorithm for job scheduling which outputs $1+\frac{1}{k}$ approximate solution in time.

poly n and k. Then what we do we put first you see that opt is less than equal to n times $q(n)$, opt is definitely less than equal to the sum of processing times of all jobs. So, is less than equal to $q(n)$. What we do we put k to be ceiling of twice n $q(n)$. Then the alg, we know alg is must be less than equal to $\left(1+\frac{1}{k}\right)\times opt$, opt is less than equal to n $q(n)$. So, $(1+\frac{1}{k})\times nq(n)$ and k is this because it is a ceiling what we can do this is less than equal to $1+\frac{1}{2nq(n)}$.

So, let us make the first term opt keep it opt plus the second term is at most $\frac{nq(n)}{k}$. So, for the first term let us keep it $opt+\frac{nq(n)}{2nq(n)}$. So, this is less than equal to $opt+\frac{1}{2}$, but we have assumed that all processing times are integral so that means, this is less than equal to opt. of course, l it is a schedule.

So, ALG must be greater than equal to opt. So, what we have obtained is opt equal to ALG. So, the algorithm outputs a schedule in which is optimal schedule and the runtime is poly n and k and k is $nq(n)$. which is because q n is a polynomial is poly n. So, in if we have a FPTAS for this job scheduling algorithm and because it is strongly NP complete, we have shown that we can use that FPTAS to get an optimal schedule when the processing times are polynomially bounded in n in time polynomial which implies that P equal to NP.

So, under P not equal to NP assumption there is no FPTAS for the job scheduling algorithm on multiple parallel machines to minimize the mix span ok. So, let us stop here. Thank you.