Approximation Algorithm

Prof. Palash Dey

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Week – 03

Lecture 12

Lecture 12 : Greedy Algorithm for Scheduling Jobs on Multiple Identical Machines

Welcome. So, in the last class we have seen a $(2-\frac{1}{m})$ factor approximation algorithm for ah scheduling jobs on multiple identical machines and we have used the local search based method for designing that algorithm. So, in this class we will see a greedy algorithm for designing ah this scheduling. So, a greedy algorithm to schedule jobs on multiple identical machines. So, we will design and we will discuss an algorithm which is called list scheduling algorithm. So, we list the jobs arbitrarily let it be $j_1,\ldots,j_n$ and in each iteration I pick a job and assign it to the least heavy machine, least busy machine.

In each iteration initially all jobs are unassigned in each iteration we pick the next job and assign it to the least heavy, least loaded machine. Clearly this is a polynomial time algorithm, we first claim that this is a two factor approximation algorithm. The above greedy algorithm has an approximation factor of at most $(2-\frac{1}{m})$. And the proof is very easy it is only the observation that if we start the local search algorithm with the output of the greedy algorithm, the local search algorithm terminates in in one iteration it does not make any local move.

If we start the local search algorithm, local search based algorithm for job scheduling with the output of the above greedy algorithm, then the local search based algorithm does not make any local move and terminates immediately. why this is so? Because for each machine look at the job which is running at the last to see this for each machine say machine j consider the job i_j that is executed last on that machine. Now, the greedy algorithm assign this job i_j to that machine means that all other machine are as where are as busy as that machine j at that time when i_j has started. So, since the greedy algorithm assigned job i_j to machine j, no machine was available at time $C_{ij}-p_{ij}-1$ at time before this ok. And in subsequent iterations other jobs got assigned to other machines in subsequent iterations of the greedy algorithm only increases the load of other machines.

Hence all machine are busy. at time $C_{ij} - p_{ij} - 1$. This implies that there is no local move for from for the job i_j. This implies there is no local move available from machine M_j. ok and because now the algorithm terminates the local search based algorithm terminates and we have already seen that the approximation ratio of the local search based algorithm is at most $(2 - \frac{1}{m})$ it follows that the approximation ratio of this greedy algorithm is also at most $(2 - \frac{1}{m})$.

Hence the approximation ratio of greedy algorithm is at most $(2 - \frac{1}{m})$ ok. So, this is the list scheduling algorithm. Next we will refine this list scheduling algorithm instead of picking or running this algorithm for arbitrary list, we will pick a specific list which is the non increasing order of the jobs. So, instead of using arbitrary list for the greedy algorithm we sort the jobs in non increasing order of their processing time and run the least scheduling algorithm. using it ok.

Now, we show that this has an approximation ratio of at most 4 by 3. So, here is a theorem. So, this is called the longest processing rule longest processing time rule. So, this algorithm is called the longest processing time rule. So, we are processing the longest jobs first.

So, there is a theorem the longest processing $\frac{4}{3}$ approximation algorithm for scheduling jobs. on identical machines to minimize makespan proof. So, it is a proof by contradiction. So, suppose not suppose not then there exists A counter example showing that the makespan of the schedule output by the algorithm is more than 4 third times the optimum makespan. There exists a counter example showing that the makespan of the schedule output by the algorithm is more than $\frac{4}{3}$ times the optimal makespan.

So, by renaming the jobs we can assume without loss of generality. So, consider such a counterexample by renaming the jobs we can assume without loss of generality that

$p_1 \geq p_2 \geq \ldots \geq p_n$

Next we say that we can assume without loss of generality that p_n is the last job. So, we can also assume without loss of generality that p_n is the last job to finish why suppose not suppose p_l is the last job if not then supposing p_l supposing p_l be the last job to finish we can delete.

the jobs $J_{l+1},\ldots,J_n$ and still the counterexample remains valid. this is so, because this is so, because deleting $J_{l+1},\ldots,J_n$ keeps ALG unchanged but possibly decreasing opportunity. So, if already with jobs $J_l,\ldots,J_n$ ALG is greater than $\frac{4}{3}$ times OPT and if I delete the jobs $J_l,\ldots,J_n$ then ALG remains same, but OPT can decrease potentially because deleting jobs can potentially decrease OPT. So, the in the new instance also ALG is greater than $\frac{4}{3}$ times OPT. So, we have assumed without loss of generality that J_n is the last job to finish ok.

Now, we say that if p_n is less than equal to $\frac{opt}{3}$, then by the analysis of the local search algorithm, then by the analysis of the local search based algorithm we have ALG. If you recall $ALG=\left(\frac{1}{m}\sum p_i\right)+p_n$ and both of them are is less than equal to opt. is less than equal to opt and this if this $p_n \le \frac{opt}{3}$ under this assumption if this is the case. Then we have this is then ALG is less than equal to $\frac{4}{3}$ opt ok. So, but this is a counter example, but we know that in this example ALG is greater than $\frac{4}{3}$ opt.

However, since it is the instance is a counter example, we have assumed that ALG is greater than $\frac{4}{3}$ opt. So, alg is so, p_n cannot be less than $\frac{opt}{3}$ because it is a counter example, hence p_n is greater than $\frac{opt}{3}$. but in this case observe that in any optimal schedule any machine cannot process more than 2 jobs. Then in the optimal schedule no machine process says more than 2 jobs ok, but it turns out that under this assumption this problem is polynomial time solvable ok. So, this this shows that this counter examples can only be polynomial time solvable.

and and in particular the greedy algorithm it is more the greedy algorithm outputs the optimal solution under this assumption the greedy algorithm outputs the optimal solution and hence this there cannot exist no such counter example. So, here is a lemma which I leave it as a homework for you to prove not difficult to prove just some simple case analysis that for any input if the processing time if the processing time of every job is more than $\frac{opt}{3}$. the greedy algorithm outputs an optimal solution. Hence, no count no such counter example can which in turn shows that the approximation factor of the greedy algorithm is at most fourth third which proves the claim ok.

Let us stop here. Thank you.