Approximation Algorithm

Prof. Palash Dey

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Week – 03

Lecture 11

Lecture 11 : Local Search Algorithm for Scheduling Jobs on Multiple Identical Machines

Welcome. So, in the last class we have seen a greedy algorithm for k center problem and we have seen its approximation ratio is 2. So, in today's class we will see local search heuristics and greedy algorithms for designing approximation algorithms and for that we will use the example of scheduling jobs in identical parallel machines. So, today's problem is scheduling jobs on identical parallel machines. So, what is input? n jobs with processing times $p_1, p_2, \ldots, p_n$ and we have m identical machines.

ok we have m identical machines and what is the goal. So, we look at the completion times of all the jobs and we want to minimize the maximum completion time. So, let C_j be the completion time of job j in a schedule and define $C^{max} = max_{j=1}^{n} C_j$, this is the maximum completion time ok. The goal or the goal is to compute a schedule which minimizes C^{max}. We assume non-preemptive scheduling that means, jobs once started should be it should be allowed to run continuously till it finishes. All the jobs are available at time 0, this is unlike the scheduling jobs on a single machine. So, all the jobs are available time 0, there are no deadline for the jobs, but finish jobs as quickly as possible. This C^{max} is also called make span of a schedule.

makespan of a schedule or maximum load of any machine. So, this problem can also be viewed from load balancing point of view, we want to minimize the maximum load of any machine. So, here we begin with local search based algorithm. Recall what is local search based algorithm? The framework is we start with any arbitrary solution and iteratively improve the solution by local moves till we can and once there is no local move which can improve the solution we output the solution. So, here also we start with an arbitrary schedule of this n jobs into m machines.

Now, we look at the job we consider a job that finishes last. be a job that finishes last and check is it possible to move that lth job to some machine ah which reduces the the maximum load and this ah in particular suppose this is some machine M here this there are various jobs are running and the last job is the lth job which finishes at time C_l. Now, it makes sense to move this job to some machine which finishes processing all its job before the start of the lth job. So, if there is a machine which is idle that means, it has finished all its assigned job at time. What is this time? This time is $C_l - p_l$, p_l is the processing time of l-th job at times earlier than $C_l - p_l$.

we move if there is a machine $M^{'}$, we move job l to machine $M^{'}$ ok. So, that is what we do and you see by that we are between M and M prime we are if we just consider M and $M^{'}$ we are reducing the maximum load. It may still be possible that there is another machine which who is occupied processing jobs till C_l time. But, in that case also we are making progress in the sense that we are reducing the number of machines with maximum load with the number of machines who are busy till C_l. So, what you observe is that each local move can only possibly decrease the make span the current make span and if the make span remains unchanged, then the number of machines with load with maximum load decreases ok.

So, now, we show that these are two factor approximation algorithm theorem. this local search algorithm has an approximation factor of 2 proof. So, when does the algorithm terminates? The algorithm terminates when there is no local So, in the algorithm terminates. So, let C_l be the make span of the schedule output by the algorithm ok. So, we break the time into two parts one is from 0 to $C_l - p_l$.

So, here are various machines and there is one machine which is which is occupied that could be more than one machine which is occupied till time C_l just pick any one job which is which finishes at time C_l and let p_l be the processing time of that job. So, this is from this is $C_l - p_l$. So, since no local move was available that is why the algorithm has terminated. all the machines are busy in the time interval 0 to $C_l - p_l$ ok. Now, what is ALG? ALG is C_l this is the make span of the schedule output by the algorithm this C_l you can write it as.

So, define S_l to be $C_l - p_l$ this time interval. So, this is $S_l + p_l$. Now, we will see two lower bounds. So, since all machines was busy from 0 to S_l executing jobs in let us this jobs called $\{J_1, \ldots, J_n\} \setminus \{J_l\}$, l-th job because this l-th job J_l is running here. We have $\sum_{i \in [n], i \neq l} p_i$ this is greater than equal to $m \times S_l$ and any algorithm in particular the optimal schedule must take $\frac{\sum_i p_i}{m}$ time.

So, let clearly $opt \geq \frac{\sum_i p_i}{m}$. So, opt is greater than equal to the average time that any machine with needs to run is this and this one and also opt is. greater than equal to any jobs processing time in particular the processing term of l-th job p_l. So, now, use this here. So, from here we can write $S_l \leq \frac{1}{m} \sum_{i \in [n], i \neq l} p_i$.

Now, from here let us come $ALG = S_l + p_l \leq \frac{1}{m} \sum_{i \in [n], i \neq l} p_i + p_l$. Now, this is by rewriting $\frac{1}{m} \sum_{i \in [n]} p_i + (1 - \frac{1}{m}) p_l$, we add and subtract $\frac{p_l}{m}$. Now, this sum is less than equal to opt $p_l \leq opt$. So, this is less than equal to opt, this is less than equal to opt. So, we have this is less than equal to $opt + (1 - \frac{1}{m}) opt$ which is $(2 - \frac{1}{m}) opt$ ok.

So, this is better than two factor approximation algorithm. Now, we have one problem here what is the running time of this algorithm? as described the algorithm runs in pseudo polynomial time So, let us argue we start with a C_{max}, we start with a with any makespan and either in after every local move either the max span drops by at least 1 or if it does not drop the number of machines which is maximally loaded which is highest which has highest load by 1. So, we can say after n local moves after every $n+1$ local moves makespan drops by at least 1. And the maximum possible make span to begin with could be $\sum p_i$ total processing times. So, the running time the number of iterations could be as high as $O(n \cdot \sum p_i)$.

Here you see the running time depends on the value of the processing times, but the input is at this numbers and this numbers are encoded in binary. So, to represent the value say p_i we need only $\log p_i$ bits. So, this running time of $O(n \cdot \sum p_i)$ this is not polynomial of the input size because the input size is $O(n \cdot \sum \log p_i)$. So, this is not a polynomial time algorithm as described these algorithms are called pseudo polynomial time algorithms which is polynomial of the values of the numbers. So, we use a very simple trick to convert it to a polynomial time algorithm.

Here you see we are while deciding the local move we are not selecting which machine to move. If there are multiple machines who are who is available in the time slot before $C_l - p_l$ I move to any machine. So, a greedy choice here is to move to a machine which is least loaded. So, instead of moving the job to any machine which is available before $C_l - p_l$, we move the job to the to the least busy machine to R it may not be unique least busy machine.

That means, if there are we move the jobs to the least busy machine if it is available before $C_l - p_l$ ok. So, whichever machine is available earliest we move that job to that machine. Now, with this change we can show that this is a polynomial time algorithm. So, here is a claim that each job is moved at most once. So, if I prove this then the number of iterations is n at most n and hence the running time is big of n times in each iteration can be executed in polynomial time.

So, the overall running time is polynomial time. So, it is the proof by contradiction. So, proof suppose not then there exists a job say j which is moved to M_1 and then M_2 ok. So, we will use this is the fact that the make span cannot increase.

So, let C_1 be the make span of the schedule just before the job j is moved to M_1. At that time at that time the load of M_1 be C_1 this is just before the job M_1 moves to C_1 moves to machine M_1 the job j moves to machine M_1. So, after moving the load of machine M_1 becomes C_1 plus p_j ok. So, now the next time the job moves to moves from M_1 to M_2, the load of M_2 just before j moves to M_2 be C_2, but because it is moved because job j is moved we have C_2.

less than C_1. Recall a job is moved from one machine to another only if it is it the last job when it is started that job is at that time another machine was available, but this contradicts our fact that the make span is non decreasing. and the greedy choice this contradicts the greedy choice ok. So, this shows that every job can move at most once and hence it is a polynomial type algorithm ok. So, let us stop here.