Approximation Algorithm

Prof. Palash Dey

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Week – 02

Lecture 09

Lecture 09 : Scheduling Jobs with Deadlines and Release Dates on a Single Machine

Welcome. So, today we will start with a new algorithm design techniques in general it is an algorithm design technique which is also useful for approximation algorithm design which are called greedy algorithms and local search. So, just write greedy algorithm and local search ok. The idea of greedy algorithm is as we have seen many times including the greedy set cover that we make greedy choice in every iteration while building solution incrementally ok. So, this is the greedy for example, in the set cover problem we were picking iteratively a set in each iteration which looks best in that iteration and we keep doing it until we have a solution. On other hand in the local search, we begin with a solution with a solution and make local changes in every iteration to improve the solution.

Both these techniques have been used extensively both for designing algorithms with probable guarantee especially with for designing heuristics they are very popular often for many problems these techniques is useful for designing heuristics which work well on practice and we here we will see the use of these techniques for designing approximation algorithms. Again as usual we will pick various problems use these techniques to design approximation algorithms thereby learning the use of these techniques. So, our first problem is scheduling jobs with deadlines. on a single machine ok.

So, what is the what is the setup? We have n jobs with release times each job is available from one particular time with release time processing times each job needs certain amount of time for processing it on that single machine, processing times p_j. p_j units of time $j=1,\ldots,n$ and deadlines $d_j, j=1,\ldots,n$. ok and we assume that the schedule starts at time 0, we assume that the schedule starts time 0 and every release times are non negative is greater than equal to 0. And, if we complete the processing of a job j at C_j time, if we complete the processing job j at time C_j then its lateness how late it is l_j is its lateness let us keep it capital is $C_j - d_j$ ok. And our goal is to design a schedule which minimizes the maximum lateness.

a schedule which minimizes maximum lateness l_j. Unfortunately, this is one can show that this problem is NP hard which is often the case. So, theorem you can prove it as a homework. it will it turns out that most of the problems that we deal in this course are NP-hard problems. So, you can show that set cover is a NP-hard problem, vertex cover is a NP-hard problem, this is an NP-hard problem.

This the decision version when you say this is an NP-hard problem, this term is loose because NP framework, NP hardness those are designed for decision problems. So, when I say that this problem is NP-hard that means, the decision version is NP-hard. What is the decision version? Everything is given instead of minimizing the maximum lateness there is a target lateness given and the question is does there exist a schedule we where the maximum lateness is less than equal to the target lateness. So, that is the decision version of the of this optimization version. problem is NP complete ok, without much without losing too much bigger let us write this.

not only this given it is even NP hard to design to see if there exist a schedule which finishes all jobs before its deadline. That means, where the maximum lateness is 0 is or less than equal to 0. So, there is another theorem it is NP hard to check to compute if max j equal to 1 to n let us call it L_{max}. if L_{max} is less than equal to 0. So, here is a problem this approximability framework ratios those things break if the optimum is negative.

So, this is one problem that is optimum is negative and it is difficult to check whether optimum is negative or not less than equal to 0 or not. And if it is 0 then that is also difficult to check and this implies that as a corollary we can write that there is no rho factor approximation algorithm for this problem. is easy to show basically assume that there exist a row factor approximation algorithm it is approved by contradiction. Now, if there is a row factor approximation algorithm using this you can decide whether L max is less than equal to 0 or not. So, when we have such a scenario we assume something on the input.

So, that you know this checking whether L_{max} is less than equal to 0 is becomes trivial in particular if we assume some structure on the input. So, that L_{max} is always positive. greater than 0, then this impossibility results is hardness results, interactability results is not directly applicable only of course, the NP hardness result is still applicable. So, for towards that what we assume is we assume this is a easy and common work around to assume that all due dates all due times d_j's are negative. this is not practically motivated it is like we are assuming something just to have an algorithm with probable guarantee which will of course, work in practice work in practice also very well.

So, this guarantees that L_{max}. maximum lateness is strictly positive. Now, we will show we will give a two factor approximation algorithm for this problem. We will give a two factor approximation algorithm in this setting assuming all due dates or due times are negative and we will use greedy algorithms. So, for that we will use a lemma which lower bounds the opt in some sense.

So, here is that important lemma which we will use crucially in algorithm design and analysis more in analysis. So, it says for each set S of jobs for each subset S of jobs, we have else max let us write it L^*_{max} which is the optimal value which is opt. $L^*_{max} \geq r(S) + p(S) - d(S)$. What are these terms? $r(S)$ is the minimum release time among the jobs in S what is the time when what is the time what is the earliest time when any job in S was available. So, this is $r(S) = min_{j \in S} r_j p(S) = \sum p_j \; d(S) = max_{j \in S} d_j$ and proof is very easy. So, consider the optimal schedule consider any optimal schedule. Optimal schedule restricted to set S it means focus only on the jobs on set S. So, let j be the job in S processed last. Now, see that because none of the jobs in S can be processed before $r(S)$ and total time required for processing all jobs is $p(S)$ it follows that ok. So, since none of the jobs in S is available before $r(S)$ and total time total processing time required for jobs in S is $p(S)$, the job j finishes on or after $r(S) + p(S)$ ok.

So, that means, and the due date of job j was $d(S)$ or earlier because $d(S)$ is the maximum due date. The due date of job j was on or before $d(S)$. So, what we have is C_j is greater than equal to $r(S) + p(S)$ ok. And lateness $L_j \geq r(S) + p(S) - d(S)$. That means, this implies that L^*_{max} can only be greater than equal to L_j which is in turn greater than equal to $L^*_{max} \geq r(S) + p(S) - d(S)$.

So, this gives a lower bound on the optimal. So, now, what is the algorithm? The algorithm is a greedy and quite obvious algorithm and that algorithm is called earliest due date first. earliest due date EDD algorithm rule. It says that among or at each moment that the machine is idle. we do not allow preemption that means, a job once started must be finished or must be allowed to run for its processing time.

At each moment that the machine is idle start processing. Next, an available job whose due date is earliest ok. So, we will show that these are two factor approximation algorithm theorem. is 2 approximation algorithm assuming this is very important $d_j < 0$ for all $j \in [n]$.

proof ok. So, let let us consider the execution of an EDD algorithm and let j be the job who is maximum late. Let j be the job with maximum lateness that is then ALG which is say $L_{max} = C_j - d_j$. So, now let us focus on the time in the schedule C_j where the job j got

finished and find the earliest time t less than C_j when the machine was was processing without idle time. So, from t to C_j no idle time, let t be minimum such the machine is never idle in this interval t to C_j. ok. Several jobs may have may have been processed in this in this in this time interval t to C_j.

So, let that set S be the set of jobs processed in $[t, C_j]$. in this interval with the choice of t we know that the time unit just before t no job was available. So, by choice of t no job was available in $(t-1)$-th time unit ok. So, now what we have is that is $r(S)$ the earliest release time of the jobs in S is t. that means, $r(S)=t$ and what is $p(S)$? $p(S)=C_j-t$ because these are the jobs that are continuously processed.

So, $p(S)=C_j-t$ and so, this is because $r(S)=t$ this $C_j-r(S)$. thus we have $C_j=r(S)+p(S)$ ok. Now, let us write L^*_{max}. which is equal to $L_j=r(S)$ $L_j=C_j-d_j$. So, this is equal to $C_j=r(S)+p(S)$.

this is greater than equal to $r(S)+p(S)-d_j$ is negative here we are using d_j is negative. So, I can drop this. So, because d_j is negative this is very important So, this is one and the other is that on the other hand also applying the lemma on the singleton job $S=\{j\}$ we have L^*_{max}. is greater than equal to $r_j+p_j-d_j$ ok. And this is because d_j is negative or anyway because r_j and p_j are positive non negative can write right this is greater than equal to $-d_j$ ok.

And here from here I can write l_j sorry this let us first compute L^*_{max} is not L_j, L_j is the algorithms algorithms lateness. So, L^*_{max} is greater than equal to this using lemma $r(S)+p(S)-d(S)$. Now, because d_S is negative this is we can write $r(S)+p(S)$ this is because $d(S)$ is negative that means, minus $d(S)$ is positive, but this is nothing, but C_j this is greater than equal C_j. So, what we have is L_{max} which is ALG this is equal to C_j-d_j . Now, C_j is less than equal to L^*_{max} and $-d_j$ is also less than equal to L^*_{max}.

this is twice L star max which concludes the proof. Thank you.