Lecture 10: October 2, 2003

Lecturer: J. van Leeuwen *Scribe: S.R. van der Schuyt*

10.1 Overview

The topic of this lecture is the problem of assigning n tasks to m persons when $n > m$. In this case, we no longer speak of tasks and persons, but rather of jobs and machines. Assigning jobs will be referred to as *scheduling.* After a short introduction to scheduling problems, we will describe a number of problems in which any machine may do any job. First we examine the problem where each job takes the same amount of time on any machine ('identical machines'). Next we examine the case in which jobs may take different amounts of time on different machines ('unrelated machines').

10.2 Introduction to scheduling problems

In scheduling problems there are a number of concerns. When describing a scheduling problem we must take them into account:

- *the properties of the jobs.* This involves e.g. their processing time, release time, deadline, and priority or weight.
- *the properties of the machines.* In this lecture we will consider
  1. identical machines, and
  2. unrelated machines, in which case jobs may take a different time on different machines,

  and we assume in both cases that the machines run *in parallel.* In other models one may e.g. require that the machines are placed in a series ('flow shop scheduling') or that the jobs must be processed by their own specified route through the set of machines ('job shop scheduling').
- *the properties of the processing.* One may consider several constraints, e.g. precedence constraints, allowed waiting times, penalties for delays (lateness), whether or not preemption is allowed, or whether we consider on-line or off-line scheduling. In this lecture no special ordering constraints are placed on the jobs.
- *the objective of the scheduling.* Usually this is to design a schedule of minimum cost, whatever the cost criterion is.

A scheduling algorithm is also called a *scheduling policy*, especially in the on-line case with e.g. stochastic processing times.

10.3 Scheduling unconstrained jobs on identical machines

We consider m machines and n jobs with respective processing times $p_1, \ldots, p_n \in \mathbb{N}_+$. These processing times are the same no matter on which machine a job is run and pre-emptions are not allowed. We assume that the machines all start at time 0 and operate at the same speed.

Definition 10.1 *The* makespan *of a schedule is the latest completion time of any job in it.*

The goal is to schedule the jobs so that the makespan of the schedule is minimized.

10.3.1 Relation to bin packing

The problem described above can be related to the problem of packing n items of sizes $p_1, \ldots, p_n \in \mathbb{N}_+$ into *bins* of a certain capacity, say B. Let $bin(I, t)$ be the minimum number of bins of capacity t needed for instance I. The problem to determine $bin(I, t)$ is called *bin packing.*

Lemma 10.2 *If $bin(I, t)$ could be computed in polynomial time then so would minimum makespan scheduling.*

Proof: Note that the relevant values for t vary between 1 and $\sum_{i=1}^{n} p_i$. As t grows, the number of bins needed is monotonically decreasing. Minimum makespan scheduling concerns the question of determining OPT, the smallest possible t such that $bin(I, t) \leq m$. OPT can be found in $O(\log(\sum_{i=1}^{n} p_i))$ calls to the routine for $bin(I, t)$, by doing a binary search for the desired minimum value of t. So, if $bin(I, t)$ could be computed in polynomial time, then so could the minimum makespan. ■

Exercise. Formulate and prove a converse to Lemma 10.2.

The given reduction is relevant because the bin packing problem is known to have efficient solutions when the object sizes are small or when otherwise only a small number of different sizes occur. In general however, no polynomial-time algorithm is known for either bin packing or minimum makespan scheduling. In particular, minimum makespan scheduling on identical machines is computationally hard!

The question then arises whether there exist good approximation algorithms for minimum makespan scheduling. We will look at two techniques for obtaining 'good' approximate solutions, namely *list scheduling* and the *LPT rule* ("Largest Processing Time first").

10.3.2 List scheduling

Consider the following situation: there are m machines and n jobs with processing times p_1, $\ldots$, $p_n \in \mathbb{N}_+$, in arbitrary order. We assume that the jobs are given in a linear list, which is equivalent to assuming that the n jobs would arrive in an on-line manner.

Consider the algorithm that schedules jobs as follows:

List scheduling

while the list is not empty **do**

begin

1. take the next job from the head of the list and delete it from the list
2. assign the job to the machine with the least amount of work assigned to it so far

end

return the schedule so obtained.

Lemma 10.3 *The list scheduling algorithm operates in $O(n \log m)$ time in the worst case.*

Proof: Taking a job from the list takes $O(1)$ time and needs to be done n times. Finding the machine with the least amount of work assigned to it can be done by maintaining a heap data structure with the total amounts of work assigned to each machine so far. Finding the machine with the least amount of work so far can be done in constant time, since it is at the top of the heap. It must be deleted from the heap, and the processing time of the new task must be added to the total work assigned to the machine so far and the new total must be inserted back into the heap. These actions cost $O(1)$, $O(1)$ and $O(\log m)$ respectively and they must each be done n times. In total, the algorithm takes $O(4n + n \log m)$ time, which is equivalent to $O(n \log m)$ time. ■

Theorem 10.4 (Graham, 1966) *List scheduling achieves a performance ratio $\leq 2 - \frac{1}{m}$.*

Proof: Let $W = \sum_{i=1}^{n} p_i$, the sum of all processing times to be accommodated. We know that the total processing time available in an optimal schedule on the machines is $m \cdot OPT$. So, $OPT \geq \frac{W}{m}$. Moreover, $OPT \geq p_k$ for every k.

Let A be the makespan of the schedule produced by the algorithm. By definition there must be a job k, with processing time p_k, that ends at the makespan time. No machine can end its task before $A - p_k$, because then job k would have been scheduled on that machine, thus reducing the makespan. So all machines are busy from time 0 through $A - p_k$. Consequently,

$$
\begin{aligned}
W &\geq m \cdot (A - p_k) + p_k \\
\Rightarrow A &\leq \frac{W}{m} + \frac{m-1}{m} p_k \\
&\leq OPT + (1 - \frac{1}{m}) OPT \\
&= (2 - \frac{1}{m}) OPT.
\end{aligned}
$$

■

Exercise. Find a set of jobs that shows that $(2 - \frac{1}{m})OPT$ is the best performance ratio the algorithm can achieve. (Hint: consider a list with $m(m-1)$ jobs of length 1 followed by one long job.)

10.3.3 The LPT rule

List scheduling can do badly if long jobs at the end of the list spoil an even division of processing times. Consider the "Largest Processing Time first" or LPT rule that works as follows. We now assume that the jobs are all given ahead of time, i.e. the LPT rule works only in the off-line situation.

> **LPT rule**
>
> **sort** the jobs in order of decreasing processing times: $p_1 \geq p_2 \geq \ldots \geq p_n$
>
> **execute** list scheduling on the sorted list
>
> **return** the schedule so obtained.

With Lemma 10.3 it easily follows that the LPT rule determines a schedule within $O(n \log n)$ time in the worst case.

Theorem 10.5 (Graham, 1969) *The LPT rule achieves a performance ratio* $\leq \frac{4}{3} - \frac{1}{3m}$.

Proof: We will construct a proof by contradiction. Assume that $\frac{4}{3} - \frac{1}{3m}$ is not a valid bound on the performance ratio. Let I be an instance with the smallest number of jobs such that the LPT schedule for I has a makespan $> (\frac{4}{3} - \frac{1}{3m})OPT$. We distinguish two cases:

Case 1. $p_n \leq \frac{OPT}{3}$. Consider a job j that finishes at the makespan A and suppose $j \neq n$. Consider the instance I' equal to I without job n. Then I' is a smaller instance than I for which the LPT rule computes a makespan $> (\frac{4}{3} - \frac{1}{3m})OPT$, because the completion time of the LPT schedule for I' is *also* equal to A and $A > (\frac{4}{3} - \frac{1}{3m})OPT_I > (\frac{4}{3} - \frac{1}{3m})OPT_{I'}$. This contradicts the choice of I. So we can assume that $j = n$. Then by an argument which is similar to the proof of Theorem 10.4 we have:

$$\begin{aligned} \Rightarrow A &\leq \frac{W}{m} + \frac{m-1}{m}p_n \\ &\leq OPT + \frac{m-1}{m}\frac{OPT}{3} \\ &= (\frac{4}{3} - \frac{1}{3m})OPT, \end{aligned}$$

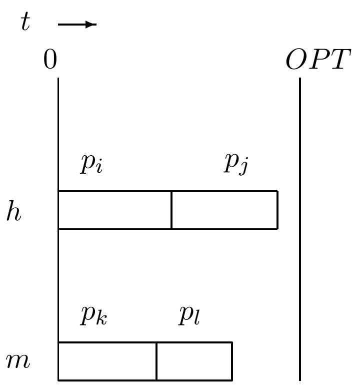
which completes the proof for *Case 1.*

Case 2. $p_n > \frac{OPT}{3}$. Consider the optimum schedule, with makespan OPT. There can clearly be no more than 2 jobs per machine. Normalize the optimum schedule as follows:

- if a machine has two jobs place the longest first,
- sort the machines so that the first jobs are in descending order of processing time.

If there are $\leq m$ jobs total, then normalize further and spread the jobs so each machine can be assumed to have at most one job. One easily sees that the LPT rule gives a schedule equivalent to this, thus with makespan $OPT < (\frac{4}{3} - \frac{1}{3m})OPT_I$, contradiction.

Thus assume that there are $> m$ jobs. This means that some machines must have 2 jobs assigned to them. Without loss of generality we may assume that every machine has two jobs assigned to it (e.g. by adding dummy jobs with processing time 0). We will again normalize the optimum schedule further. Let the first, thus longest job of machine m be the job with processing time p_k. Note that this job is the shortest of the jobs in first place on any machine. Suppose there is another machine h with first job p_i and second job p_j, such that $p_j > p_k$.



We can interchange jobs j and k and keep an optimal schedule! For, $p_k < p_j$ so $p_i + p_k \leq OPT$ on machine h. On machine m we have jobs p_j and p_l. Necessarily $p_j + p_l \leq OPT$, because $p_i + p_j \leq OPT$ and $p_l \leq p_k < p_j \leq p_i$.

By iterating exchanges of this kind, it follows that there is an optimum solution in which the m biggest jobs are all in first positions on the machines, all remaining (smaller) jobs are in second positions, and moreover all these remaining jobs appear from machine m down to machine 1 in order of decreasing size.

One easily sees that the LPT rule gives a schedule equivalent to this, thus with makespan $OPT < (\frac{4}{3} - \frac{1}{3m})OPT_I$. This contradicts the assumption. So we have concluded the proof for *Case 2* also. ∎

10.4 Approximation scheme for identical machines

Several other polynomial-time scheduling policies exist for which the performance ratio can be bounded by a 'small' constant. In fact, for every fixed m and every $\varepsilon > 0$ there is a polynomial-time algorithm that computes an approximately optimal schedule with a performance ratio of $\leq 1 + \varepsilon$.

In order to show this, we design a rule that trades 'computation time for quality'. Let c be an arbitrary but fixed integer constant ≥ 1.

LPT$_c$ rule

sort the jobs in order of decreasing processing times: $p_1 \geq p_2 \geq \ldots \geq p_n$

assign the first c jobs according to an *optimal* schedule

and continue scheduling

execute list scheduling on the remaining part of the sorted list

return the schedule so obtained.

An optimal schedule for jobs $1, \ldots, c$ may be determined by enumerating all possibilities, thus in $O(n^c)$ time. For any fixed constant c, the LPT$_c$ rule is polynomial-time computable.

Lemma 10.6 (Horowitz and Sahni, 1976) *The LPT$_c$ rule achieves a performance ratio* $\leq 1 + \frac{m-1}{c+1}$.

Proof: Say the optimal schedule of the first c jobs has makespan B. Let the overall makespan resulting after placing the remaining $n - c$ jobs be A. If $A = B$, then obviously $A = OPT$ and we are done. Thus assume that $A > B$.

Let p_k be the last job to finish, necessarily at time A and necessarily also $k > c$ (because the first c jobs are all finished by time B). At the moment job k was assigned, no machine could have his load end before time $A - p_k$ otherwise job k would have been assigned to this machine and end earlier. Thus all machines are at least fully busy up to time $A - p_k$, with jobs $1, \ldots, k-1$. Thus:

$$A - p_k \leq \frac{p_1 + \ldots + p_{k-1}}{m} = \frac{p_1 + \ldots + p_k}{m} - \frac{p_k}{m} \leq OPT - \frac{p_k}{m}$$

and hence

$$A - OPT \leq (1 - \tfrac{1}{m}) p_k \leq (1 - \tfrac{1}{m}) p_{c+1}.$$

On the other hand, observe that $OPT \geq \frac{1}{m}(p_1 + \ldots + p_{c+1}) \geq \frac{c+1}{m} p_{c+1}$, thus $p_{c+1} \leq \frac{m}{c+1} OPT$. It follows that

$$A \leq OPT + (1 - \tfrac{1}{m}) p_{c+1} \leq (1 + (1 - \tfrac{1}{m}) \tfrac{m}{c+1}) \cdot OPT = (1 + \tfrac{m-1}{c+1}) OPT,$$

and the bound on the performance ratio follows. ■

Suppose m is *fixed* and let an $\varepsilon > 0$ be given. If c is chosen such that

$$1 + \frac{m-1}{c+1} \leq 1 + \varepsilon$$

thus $c \geq \lceil \frac{m}{\varepsilon} \rceil$, then the LPT$_c$ rule is a 'polynomial-time' scheduling policy with a performance ratio $\leq 1 + \varepsilon$!

Interestingly, a different technique shows that the assumption that m be fixed can be *removed*.

Theorem 10.7 (Hochbaum and Shmoys, 1987) *There exists an algorithm scheme $A(I, \varepsilon)$ such that for any fixed $\varepsilon > 0$, $A(I, \varepsilon)$ computes an approximate schedule for minimum makespan scheduling on identical machines with performance ratio $\leq 1 + \varepsilon$ and runs in time polynomial in $|I|$.*

10.5 Scheduling on unrelated machines

Consider m parallel machines and n jobs (*cf* section 10.2). Let M be the set of machines and J the set of jobs. This time we are given processing times $p_{ij} \in \mathbb{N}_+$, with

$$p_{ij} = \text{the processing time/costs of running job } j \text{ on machine } i.$$

We can model the minimum makespan scheduling problem in this case as a 0-1 LP as follows:

min $\quad t$

subject to

$$\begin{aligned} &\textstyle\sum_{i \in M} x_{ij} = 1 && \text{for all } 1 \leq j \leq n \\ &\textstyle\sum_{j \in J} p_{ij} x_{ij} \leq t && \text{for all } 1 \leq i \leq m \\ &x_{ij} \in \{0, 1\} && \text{for all } 1 \leq i \leq m,\ 1 \leq j \leq n. \end{aligned}$$

We will show that a suitably relaxed family of LP's can provide the means for computing a 'fairly good' approximate schedule.

10.5.1 Relaxing the LP model

Now consider the ordinary relaxation of the LP model, replacing $x_{ij} \in \{0, 1\}$ by $x_{ij} \geq 0$. This may lead to fractional solutions, i.e. split jobs in the schedule. This is not the way to go, because this cannot easily be modified to a good solution in which jobs are not split. Especially long jobs that get reassembled from their split parts when we round back to the 0-1 LP may unbalance the schedule.

Use time thresholds t with t a positive *integer* and let us look at the sets:

$J_i(t) = \{j | p_{ij} \leq t\}$, and

$M_j(t) = \{i | p_{ij} \leq t\}$,

and do this for those thresholds t for which all jobs are still present in $J_i(t)$, i.e. for all $t \geq t_0$ where t_0 is the smallest time such that $\bigcup_{i \in M} J_i(t_0) = J$. With the parameter t we can filter out the 'large' processing times, implicitly setting $x_{ij} = 0$ when $p_{ij} > t$.

Thus look at the 'alternatively relaxed' models LP_t defined by:

$$\min \quad 1$$

subject to

$$\sum_{j \in J_i(t), i \in M_j(t)} x_{ij} = 1 \qquad \text{for all } 1 \le j \le n$$

$$\sum_{j \in J_i(t), i \in M_j(t)} p_{ij} x_{ij} \le t \qquad \text{for all } 1 \le i \le m \text{ with } J_i(t) \neq \emptyset$$

$$x_{ij} \ge 0 \qquad \text{for all } j \in J_i(t), i \in M_j(t).$$

Minimizing a trivial function like '1' is enough when we are only interested in whether the constraints have a feasible solution or not. Consider *any* threshold value $t \ge t_0$ such that LP_t has a feasible solution, i.e. a time within which all jobs can be scheduled using their p_{ij}-values $\le t$. Clearly all $x_{ij} \le 1$, by the first constraint.

Lemma 10.8 *An extreme point of the LP polytope of LP_t has $\le m + n$ coefficients $\neq 0$.*

Proof: The extreme points of the polytope are determined by the 'basic solutions' of the system of linear equations corresponding to the constraints. This system is of the form $Ax = b$ where A has $\le m + n$ rows, thus rank $\le m + n$. Basic feasible solutions are known to have at most $rank(A)$ coefficients $\neq 0$. ■

Lemma 10.9 *In any extreme point solution x of LP_t there will be at least $n - m$ coefficients equal to 1.*

Proof: Consider any solution $x = (\ldots, x_{ij}, \ldots)$, in which at most $n + m$ of the x_{ij}s are not equal to 0 (cf. the previous Lemma). Say α jobs have $x_{ij} = 1$ and β jobs have $0 < x_{ij} < 1$, out of the less than $n + m$ coefficients that are $\neq 0$. This leads to the following (in)equalities:

$$\alpha + \beta = n$$

$$\alpha + 2\beta \le n + m$$

where the latter follows because each of the β 'fractional' jobs is apparently split over at least 2 machines and thus gives rise to at least 2 non-zero coefficients in x. Solving for α we conclude: $\beta \le m$ and $\alpha \ge n - m$. ■

10.5.2 An approximately optimal scheduling policy

How does the information about the LP_t help? We will design an approximation algorithm and show the following result.

Theorem 10.10 (Lenstra, Shmoys, and Tardos, 1987) *The minimum makespan scheduling problem for unrelated machines can be solved by an efficient algorithm within performance ratio ≤ 2.*

Proof: Design an approximation algorithm as follows. Determine the *smallest* threshold value T such that the problem LP_T has a feasible solution (e.g. by binary search). Obviously $T \leq OPT$.

Take an extreme point solution $x = (x_{ij}$ of LP_T. Assign all jobs j that have an $x_{ij} = 1$ in the extreme solution to machine i. The length ('duration') of this assignment is $\leq T \leq OPT$.

However, we are still stuck with the fractional jobs, i.e. jobs which are assigned partially to one machine and partially to one or more other machines. Remember that the related $p_{ij} \leq T \leq OPT$. We can model this situation as a bipartite graph G. There is one vertex for each fractional job and one for each machine that occurs with a fractional x_{ij}. For each fractional assignment x_{ij} in the solution we add an edge in G between machine i and job j.

Fact 10.11 *G has a perfect matching.*

This fact gives us a way of assigning jobs to machines! Following the perfect matching, at most one further job of duration $\leq T \leq OPT$ is added to each machine. This gives a valid schedule with a makespan $\leq OPT + OPT$. Thus the performance ratio is 2. ■

Theorem 10.12 (Lenstra, Shmoys, and Tardos, 1987) *There does not exist a polynomial-time approximation algorithm for the minimum makespan scheduling problem on unrelated machines with a performance ratio $< \frac{3}{2}$, unless $P = NP$.*

References

[1] R.L. Graham. Bounds for certain multiprocessing anomalies, *Bell System Techn. Journal* 45 (1966) 1563-1581.

[2] R.L. Graham. Bounds on multiprocessing timing anomalies, *SIAM J. of Applied Mathematics* 17 (1969) 263-269.

[3] D. S. Hochbaum, D.B. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results, *Journal of the ACM* 34 (1987) 144-162.

[4] E. Horowitz, S.K. Sahni. *Fundamentals of computer algorithms.* Pitman Publishing Company, London, 1978.

[5] J.K. Lenstra, D.B. Shmoys, E. Tardos. Approximation algorithms for scheduling unrelated parallel machines, in: *28th. Annual IEEE Symposium on Foundations of Computer Science*, IEEE Computer Society Press, 1987, pp. 217- 224.

[6] V.V. Vazirani. *Approximation algorithms*, Springer-Verlag, Berlin, 2001.