Scheduling Theory **CO 454, Spring 2009**

Homework 5

Due: July 21st, 2009

(Total 30)

1. (6 marks)
   State if the following statements are true or false. If true, prove it. If false, give a counter-example.

   - **(3)** *If the number of jobs $n < 2m$, then LPT returns an optimal solution to* $(P||C_{max})$.
     Solution: False. Consider the instance with $m = 4$ machines, and $n = 7$ jobs with times $(6, 6, 3, 3, 2, 2, 2)$. LPT gives a schedule of makespan 7, and the optimum is 6.
   - **(3)** *If the number of jobs $n > km$, then LPT returns a $\frac{k+1}{k}$ factor approximation algorithm.*
     Solution: False. Consider the instance with $m = 2$ machines and 15 jobs – 5 jobs have times $(300, 300, 200, 200, 190)$ and 10 jobs with processing times 1. Thus with $k = 7$, $n > km$. If the above were true, LPT should have returned a 8/7 factor algorithm.

     Optimum is 600, which processes the first two on one machine, and the remaining 13 on the second machine. LPT returns a schedule with makespan 690. $690/600 > 8/7$ falsifying the statement.

2. (8 marks)
   Consider the following generalization of list-scheduling for minimizing makespan in unrelated machines $(R||C_{max})$.

   > *Let $J = \{1, 2, \ldots, n\}$ be an arbitrary ordering of the jobs and process the jobs in this order. When job j is being processed, schedule it on the machine on which it will finish the first, that is, the one which minimizes C_j. Thus, if $load_j(i)$ is the sum of processing times of all jobs from $\{1, \ldots, j\}$ assigned to machine i, then schedule job j on the machine which minimizes*
   >
   > $$load_j(i) + p_{ij}$$
   >
   > *Note that $C_{max} = \max_i load_n(i)$.*

   - **(2)** *Run the above algorithm on the following instance of $(R||C_{max})$. The entry in the ith row and jth column is p_{ij}. Assume the order is $(1, 2, 3, 4)$.*

	Job 1	Job 2	Job 3	Job 4
M/c 1	1	2	4	4
M/c 2	3	1	3	4
M/c 3	2	3	4	3

     Solution: In this order, the jobs are scheduled as follows. Job 1 goes to machine 1, Job 2 to machine 2, Job 3 to Machine 2, and Job 4 to machine 3. Makespan is 4 on machine 2.

     (2) *Does this return an optimal schedule? Is there any order such that running the above algorithm on that order will lead to an optimal schedule?*
     Solution: No, this is not optimal. The optimal schedule processes jobs 1 and 2 on machine 1, job 3 on machine 2 and job 4 on machine 3 to get optimal makespan of 3.

     If the order is $(4, 3, 2, 1)$, then the above algorithm returns the optimal solution.

- **(4)** *Show that for any constant $B > 1$, there is an example of an instance and an order of the jobs such that the above algorithm returns a schedule with makespan $C_{max} \geq B \cdot OPT$. That is, the above algorithm is not a constant factor algorithm.*

  Solution: Consider the instance with $(2B+1)$ jobs $j_1, \ldots, j_{2B+1}$ and $2B+1$ machines $1, \ldots, 2B+1$. The processing times are as follows. Let $\epsilon \leq 1/B$. Job j_1 takes time $1+\epsilon$ on machine 1 and ∞ on all other machines. Job j_2 takes time 1 on machine 1 and time 2 on machine 2, and ∞ on all others. Job j_3 takes time $1+\epsilon$ on machine 2 and time 3 on machine 3, and ∞ on all others. In general, for $r > 2$ job j_r takes time $1+\epsilon$ on machine $r-1$ and time r on machine r, ∞ on all others.

  The optimum solution is to schedule jobs j_1 and j_2 on machine 1, and job j_r for $r \geq 3$ on machine $(r-1)$. The makespan is $2+\epsilon$.
  If the order is $(1, 2, 3, \ldots, 2B+1)$, then job 1 will be processed on machine 1, job 2 will be processed on machine 2 (since it completes earlier there), job 3 on machine 3, and in general job r on machine r, giving a makespan of $2B+1$. Thus, $C_{max} = 2B+1 \geq B(2+\epsilon) = B \cdot OPT$.

3. (10 marks)
In this question, we will develop a $(2 - \frac{1}{m})$-algorithm for makespan minimization in uniformly related machines ($Q||C_{max}$). Recall, in this problem, jobs have processing time $(p_1, \ldots, p_n)$ and machines have speeds $(s_1, \ldots, s_m)$, and the time taken by machine i to process job j is $\frac{p_j}{s_i}$. The goal is to minimize the completion time of the last job.

Suppose we order the machines in decreasing order of speeds: $s_1 \geq s_2 \geq \cdots \geq s_m$. Consider the following extension of the longest processing time rule:

> *Order the jobs in decreasing processing time order. $p_1 \geq p_2 \geq \cdots \geq p_n = p_{min}$. In this order, schedule job j on the machine in which it will finish the first. Thus, if $load_j(i)$ is the sum of processing times of all jobs from $\{1, \ldots, j\}$ assigned to machine i, then schedule job j on the machine which minimizes*
>
> $$\text{load}_j(i) + \frac{p_j}{s_i}$$

(a) **(1)** *Suppose the optimum schedule doesn't assign any job to machine i. Argue that the optimum schedule doesn't assign any jobs to machines $i+1, i+2, \ldots, m$, either.*

Solution: If the optimum solution assigned any job to machine i' with $i' > i$, then moving all those jobs to machine i will decrease their completion times since i is faster than i'.

(b) **(1+2)** *(Lower Bounds) Following part (a), suppose the optimum schedule assigns jobs to machines $(1, 2, \ldots, k)$. Show that*

- $OPT \geq \frac{p_{min}}{s_k}$
- $OPT \geq \frac{\sum_{j=1}^{n} p_j}{\sum_{i=1}^{k} s_i}$

Note that these generalize the lower bounds which we had for $(P||C_{max})$.

Solution: Since machine k processes at least one job, and that job has processing time at least p_{min}, the time taken by machine k is at least $\frac{p_{min}}{s_k}$.

Let $J_1, J_2, \ldots, J_k$ be the set of jobs processed by machines 1 to k. Let $p(J_i) := \sum_{j \in J_i} p_j$. Note that

$$OPT = \max_i \frac{p(J_i)}{s_i}$$

Therefore,

$$OPT \geq \frac{\sum_{i=1}^{k} \sum_{j \in J_i} p_j}{\sum_{i=1}^{k} s_i} = \frac{\sum_{j=1}^{n} p_j}{\sum_{i=1}^{k} s_i}$$

(c) **(1)** *As done for the analysis of LPT, argue that we can assume the last job to finish is n, the job with the minimum processing time.*

Solution: If the last job j is not the job n, then consider the instance with jobs $(1, 2, \ldots, j)$. LPT returns the same schedule while optimum decreases. If we can prove LPT is a factor 2 in this instance, then $LPT(1, \ldots, n) = LPT(1, \ldots, j) \leq 2OPT(1, \ldots, j) \leq 2OPT(1, \ldots, n)$, implying LPT is a factor 2 in the original instance as well.

(d) **(2)** *Let ℓ be the machine which processes job n. Let t_n be the time when job n was started on machine ℓ. Let $load(i)$ be the total time taken by jobs from 1 to $n-1$ on machine i. Show, using the definition of the algorithm,*

$$\text{load}(i) \geq t_n + \frac{p_n}{s_\ell} - \frac{p_n}{s_i}$$

Solution: Since job n is processed on machine ℓ, it must be by the running of the algorithm that ℓ minimizes

$$\text{load}(i) + \frac{p_n}{s_i}$$

over all machines i, and therefore,

$$\text{load}(\ell) + \frac{p_n}{s_\ell} \leq \text{load}(i) + \frac{p_n}{s_i}$$

for all i. The proof is complete by noting that $\text{load}(\ell) = t_n$.

(e) **(3)** *Show, using the upper bounds of part (b) and using part (d), that the completion time of job n,*

$$C_n = t_n + \frac{p_n}{s_\ell} \leq (2 - \frac{1}{m})OPT$$

Solution: Note that $\sum_{i=1}^{k} s_i \times \text{load}(i)$ is the total processing times of all jobs processed in machines 1 to k by the algorithm. This is at most $\sum_{j=1}^{n} p_j - p_n$. This gives us

$$\begin{aligned}
\sum_{j=1}^{n} p_j - p_n &\geq \sum_{i=1}^{k} s_i \times \text{load}(i) \\
&\geq \sum_{i=1}^{k} s_i(t_n + \frac{p_n}{s_\ell} - \frac{p_n}{s_i}) = (t_n + \frac{p_n}{s_\ell})(\sum_{i=1}^{k} s_i) - \sum_{i=1}^{k} s_i \frac{p_n}{s_i} \\
&= C_n(\sum_{i=1}^{k} s_i) - kp_n
\end{aligned}$$

Rearranging,

$$C_n \leq \frac{\sum_{j=1}^{n} p_j}{\sum_{i=1}^{k} s_i} + (k-1)\frac{p_n}{\sum_{i=1}^{k} s_i}$$

Now, from part (b), we get the first term is less than OPT, and that the second term, since $s_1 \geq \cdots \geq s_k$, can be written as

$$\frac{k-1}{k} \frac{kp_n}{\sum_{i=1}^{k} s_i} \leq (1 - \frac{1}{k})\frac{kp_n}{ks_k} \leq (1 - \frac{1}{k})OPT$$

Adding, we get

$$C_n \le (2 - \frac{1}{k})OPT \le (2 - \frac{1}{m})OPT$$

since $k \le m$.

4. (6 marks)
Consider the following graph balancing *problem. We are given a undirected graph $G = (V, E)$ with weights $w(e)$ on the edges.* Orienting *an edge (u, v) of the graph is to make the edge directed, either as $(u \to v)$ or $(v \to u)$. A complete* orientation *of the edges in E leads to a digraph $D = (V, A)$ where A are the oriented arcs.*

The objective is to find an orientation which minimizes the maximum weighted in-degree *of a vertex. That is, given a vertex v, let E_v be the set of edges incident on v which have been oriented towards v. Then, the goal is to minimize*

$$\max_{v \in V} \sum_{e \in E_v} w(e)$$

Cast this problem as a problem of minimizing makespan in unrelated machines ($R||C_{max}$), and therefore argue that there is a 2-approximation for the problem.

Solution: Think of the vertices as machines and edges as jobs. Each job (edge) can go to exactly two machines (the end points) and has processing time $w(e)$ on both of them. If job (u, v) (edge) is assigned to machine (vertex) v, orient the edge towards v. Thus a schedule leads to an orientation. The minimum weighted in-degree of the orientation is precisely the makespan of the schedule.

Conversely, by a similar argument, given an orientation we can get a schedule whose makespan equals the weighted in-degree.