Lecture 5: Single Machine Environments (Maximum Lateness)

May 19th, 2009

---

1 Minimizing Maximum Lateness

In this lecture we will focus on minimizing maximum lateness in single machine environments. Recall that given a schedule and due dates, the lateness of job j is given by $L_j = C_j - d_j$. Therefore, the problems of minimizing sum (or weighted sum) of lateness is equivalent to the problem of minimizing the sum (weighted sum) of the completion times which was the focus in the last lecture. We first look at the problem with no restrictions.

1.1 $(1||L_{max})$

Since we want to decrease the maximum lateness, it makes sense to process the jobs with earlier due dates faster. In the case of this problem, this algorithm is indeed exact.

Definition 1.1. (EDD) Earliest Due-Date: Schedule jobs in increasing order of their due dates.

Theorem 1.2. *EDD gives an optimal solution to* $(1||L_{max})$.

Proof. Once again we will apply the interchange argument we saw in the last lecture. Suppose there is a schedule such that there exists jobs k and ℓ such that k is processed before ℓ and $d_k > d_\ell$. As we saw last lecture, we may assume k and ℓ are consecutive. We now argue that swapping the jobs k and ℓ will give us a schedule with no larger L_{max}. This will show that there is an optimal schedule with all d_k's nondecreasing implying EDD gives an optimal schedule.

To see this we show that in the schedule S' with k and ℓ swapped, the quantity $\max\{L_k^{S'}, L_\ell^{S'}\} \leq \max\{L_k^S, L_\ell^S\}$. Since the completion time, and hence the lateness, of any other job remains the same we will be done. Suppose in schedule S, the job j is scheduled at time t. Then $L_k^S = t + p_k - d_k$ and $L_\ell^S = t + p_k + p_\ell - d_\ell$ and thus $L_\ell^S > L_k^S$ since $d_\ell < d_k$. On swapping, $L_\ell^{S'} = t + p_\ell - d_\ell < L_\ell^S$ and $L_k^{S'} = t + p_k + p_\ell - d_k < L_\ell^S$ since $d_\ell < d_k$. Thus, the maximum is less than L_ℓ^S finishing the proof. □

2 $(1||f_{max})$

We now generalize the problem of minimizing maximum lateness as follows. Suppose we are given n nondecreasing functions $f_1, \ldots, f_n$, one for each job. Given a schedule S, the cost contributed by job j towards this schedule is $f_j(C_j^S)$. The problem $(1||f_{max})$ is to minimize

$$\max_{j \in J} f_j(C_j)$$

To see that this generalizes the problem of minimizing maximum lateness, look at functions $f_j(C_j) := C_j - d_j$.

The algorithm we describe for $(1||f_{max})$ is a little different from all the algorithms we have seen so far (with the possible exception of $(1|r_j|\sum C_j)$. All the algorithms till now were as follows: we set up a *priority rule* over all jobs, for instance, the priority rule was either the processing time, the ratio of weight to processing time, the due date; and the schedule was to process the "next available job" in the priority order on the machine. Once this priority order was set, the algorithm was purely local, in the sense it needed to know only the next item in the order and nothing else in the data. The algorithm we describe now is a little more sophisticated and takes more time than the priority-rule algorithms. However, it makes up in the generality.

Algorithm `Least-Cost-Last` (LCL)

1. Initialize $X = J$. Initialize stack S to be empty.
2. Until X is empty do
   - Find $j \in X$ which minimizes
   $$j = \arg\min_{k \in X} f_j(\sum_{k \in X} p_k)$$
   - Append j to the front of the stack S and delete j from X.
3. Schedule the jobs in the order S.

The intuition of the algorithm is as follows. We know that any optimal algorithm will not have idle time on the machine (Note that there are no side constraints). Thus, the last job j, whichever it is, faces a cost $f_j(\sum_{k \in J} p_k)$. The algorithm picks this *last* job to be the one with the *least cost*, deletes j from the set of unordered jobs, and repeats.

Before we prove the optimality, it is instructive to note for the special case of L_{max}, the LCL algorithm coincides with the EDD algorithm.

Theorem 2.1. *LCL returns an optimal schedule for* $(1||f_{max})$.

Proof. Consider an optimum schedule S which is not obtained by LCL. Suppose the order of jobs scheduled by S is $\{1, 2, \ldots, n\}$. Since S is not an LCL schedule, S schedules a job ℓ such that ℓ does not minimize $f_l(\sum_{j \in X} p_j)$ where X is the set of items scheduled before C_ℓ^S. Suppose k is the job in X such that $f_k(\sum_{j \in X} p_j)$ is minimum. Now, instead of the *interchange* operation we have seen so far, we do a *shift* operation to obtain a new schedule.

Consider the schedule S' which processes the jobs 1 to $k-1$, then processes jobs $k+1$ to ℓ, and then processes k. Thus, it has delayed the completion time of one job, k, and decreased all the completion times of jobs from $k+1$ to ℓ by p_k. Since $C_j^{S'} \leq C_j^S$ for all j except k, we know the same is true for $f_j(C_j^{S'})$. Moreover,

$$f_k(C_k^{S'}) = f_k(\sum_{j \in X} p_j) \leq f_\ell(\sum_{j \in X} p_j) = f_\ell(C_\ell^S)$$

Thus, $f_{max}^{S'} \leq f_{max}^S$. This proves that there is an optimal schedule which is LCL. □

Exercise 2.2. *What is the running time of the above algorithm?*