Lecture 4: Single Machine Environments (Release Dates)

May 14th, 2009

---

1 Minimizing Average Completion Time with Release Dates

Consider the case when there are release dates r_j for every job and we want to minimize the average completion time. That is, we want to solve $(1 \mid r_j \mid \sum C_j)$. Our first attack will be to apply the SPT rule. However, we are faced with a problem since the job with the shortest processing time might not be released at time 0. There are two possible ways we could modify SPT.

1. Sort the jobs in increasing processing times. Process jobs in this order, if job $j-1$ is finished at time C_{j-1} and $C_{j-1} < r_j$, then remain idle till r_j and process job j at time r_j.

2. Starting from time $t = 0$, at time t process the job j which has the shortest processing time among all jobs which have been released by time t; if no such job, then remain idle.

Unfortunately, none of the above algorithms are optimal. In fact, the problem is NP-hard, a concept which we will dig deeper into in the coming lectures. For the time being suffice to know that we do not expect any polynomial time exact algorithm. However, if preemption is allowed, then the above problem can indeed be solved in polynomial time.

1.1 Release Dates with Preemption $(1 \mid r_j, pmtn \mid \sum C_j)$

If preemption is allowed, then the following generalization of SPT works.

Definition 1.1. (SRPT) Shortest Remaining Processing Time: At any point of time, schedule the job with the shortest remaining processing time, preempting when jobs with shorter processing times are released.

Theorem 1.2. *SRPT gives an optimal algorithm for* $(1 \mid r_j, pmtn \mid \sum C_j)$.

Proof. The proof also follows by an interchange argument, however in this case we do not interchange two jobs but possibly parts of the two jobs. Consider an optimal schedule S in which a job k is being processed at time t while there exists an unfinished job l such that $r_\ell \leq t$, that is, the job l is available at time t, and $x_k > x_\ell$, where x_k and x_ℓ are the remaining amounts of the two jobs.

Note that the machine processes job k and l for $x_k + x_\ell$ units after time t. Construct a new schedule S' by processing only job ℓ on the first x_ℓ units of these times, and processing job k on the remaining x_k units. Since $x_\ell < x_k$, in the new schedule job ℓ finishes strictly *before* the time when either job k or ℓ finished in the old schedule. That is $C_\ell^{S'} < \min\{C_k^S, C_\ell^S\}$.

Also note that $C_k^{S'} = \max\{C_k^S, C_\ell^S\}$ and $C_j^{S'} = C_j^S$ for all other $j \neq k, \ell$. Thus, $\sum C_j$ strictly decreases which is a contradiction. □

Unlike the proof without release times, the above argument *does not* extend to the weighted case (Can you see why?). In fact, the weighted case is NP-hard even with preemption. Now we show how the above algorithm can be used to give an efficient *approximation* algorithm for $(1|r_j|\sum C_j)$.

1.2 Release Dates without preemption $(1|r_j|\sum C_j)$

Firstly, let us define what an approximation algorithm is.

Definition 1.3. Given a scheduling problem $(\alpha|\beta|\gamma)$ a ρ-approximation algorithm (for $\rho \geq 1$) returns a schedule S such that $\gamma(S) \leq \rho \cdot \gamma(S^*)$ where S^* is the optimal schedule.

We now describe a 2-approximation algorithm for the problem $(1|r_j|\sum C_j)$. This algorithm will use the exact algorithm for the problem $(1|r_j, pmtn|\sum C_j)$ to get the schedule. Henceforth we will denote the optimal schedule for the $(1|r_j|\sum C_j)$ as S^* and denote the average completion time of S^* as OPT.

Let P be the optimal schedule found by the SRPT algorithm for the problem $(1|r_j, pmtn|\sum C_j)$. Note that

$$OPT \geq \sum_j C_j^P \tag{1}$$

This is because any feasible schedule for $(1|r_j|\sum C_j)$ is a feasible schedule for $(1|r_j, pmtn|\sum C_j)$. Thus the average completion time of P is a *lower bound* on that of the optimum schedule. In fact in the schedule S that we construct, we will show that the average completion time of S is at most twice the average completion time of P rather the twice the optimum. This method of comparing the solution with a lower bound on which we have a better handle on, rather than the optimum is very prevalent in the field of approximation algorithms. We now state the algorithm.

Algorithm `Convert-From-Preemption`

1. Apply SRPT to obtain the optimal schedule P for $(1|r_j, pmtn|\sum C_j)$.
2. Order the jobs in increasing order of completion time in P, that is, $C_1^P < C_2^P < \cdots < C_n^P$.
3. Process the jobs *non-preemptively* in this order to get schedule S.

Theorem 1.4. *Algorithm* `Convert-From-Preemption` *is an efficient factor* 2*-approximation algorithm for* $(1|r_j|\sum C_j)$.

Proof. We know that $\sum_j C_j^P \leq OPT$. If we show that for every job j, $C_j^S \leq 2C_j^P$, then we will be done. Let t_j^S be the time when job j is started in schedule S. Say that the machine is idle at time t in S if it is not processing any job at time t. Note that a machine is idle only if the next job in the order to be processed on the machine has not yet been released.

Let $\texttt{idle}(t)$ denote the total time the machine is idle in schedule S until time t. Now note that

$$C_j^S = \sum_{i \leq j} p_i + \texttt{idle}(t_j^S)$$

The above is true for any non-preemptive schedule. Also note that $\sum_{i \leq j} p_j \leq C_j^P$ for by definition of the order, all jobs $i \leq j$ finish before j in P. Now comes the crucial claim.

Claim 1.5. *In the schedule S, the machine has no idle time from C_j^P to t_j^S.*

Proof. Suppose the machine is idle at time $t \in [C_j^P, t_j^S]$. Then it must be awaiting the release of some job k which comes in order before j, that is $r_k > C_j^P$. However, the reason it is in order before j is because $C_k^P < C_j^P$. In other words, P completes k before C_j^P and in particular, the job must have been released by time C_j^P. □

The above claim implies that $\texttt{idle}(t_j^S) = \texttt{idle}(C_j^P) \leq C_j^P$. Thus, we get $C_j^S \leq 2C_j^P$. □

A factor ρ approximation algorithm is *tight* if there is an example where the optimum and the schedule found by the algorithm is indeed away by a factor ρ. Is the above factor 2 approximation algorithm tight?