Scheduling Theory **CO 454, Spring 2009**

Lecture 3: Reductions among Scheduling Problems; Single Machine Environments

May 12th, 2009

---

1 Reductions among Scheduling Problems

We say a scheduling problem $(\alpha \mid \beta \mid \gamma)$ can be reduced to another scheduling problem $(\alpha' \mid \beta' \mid \gamma')$ if an algorithm for the latter problem can be used to solve the former problem. We denote this using the following notation

$$(\alpha \mid \beta \mid \gamma) \trianglelefteq (\alpha' \mid \beta' \mid \gamma')$$

The reduction is efficient if a polynomial time algorithm for the latter problem leads to a polynomial time algorithm for the former problem.

Sometimes these reductions are easy. For instance, it is easy to see that $(\alpha \mid \beta \mid \sum C_j) \trianglelefteq (\alpha \mid \beta \mid \sum w_j C_j)$, or any other non-weighted to weighted reduction. Or, $(P \mid \beta \mid \gamma) \trianglelefteq (Q \mid \beta \mid \gamma) \trianglelefteq (R \mid \beta \mid \gamma)$. However sometimes these reductions can be tricky.

Theorem 1.1. $(\alpha \mid \beta \mid L_{max}) \trianglelefteq (\alpha \mid \beta \mid \sum U_j)$

Proof. Suppose we have an algorithm to solve any instance of $(\alpha \mid \beta \mid \sum U_j)$. Consider an instance of $(\alpha \mid \beta \mid L_{max})$ with due dates $(d_1, \ldots, d_n)$ for the jobs. The crux of the theorem is in the following claim.

Claim 1.2. *Let z be the minimum value such that the instance $(\alpha \mid \beta \mid \sum U_j)$ with due dates $(d_1 + z, \ldots, d_n + z)$ has a schedule S with optimum $\sum U_j = 0$. Then S is an optimal schedule for $(\alpha \mid \beta \mid L_{max})$ with maximum lateness z.*

Proof. Since S is such that no job is late with due dates $(d_1 + z, \ldots, d_n + z)$, the completion time of every job is $C_j \leq d_j + z$. Furthermore, since z is the minimum, there must be a job j with $C_j = d_j + z$. Therefore, when the due dates are $(d_1, \ldots, d_n)$, the maximum lateness of S is $L_{max} = z$.

Now consider any schedule S' for the instance $(\alpha \mid \beta \mid L_{max})$ with due dates $(d_1, \ldots, d_n)$. Suppose the lateness is $z' < z$. By a similar argument as above, the same schedule S' with due dates $(d_1 + z', \ldots, d_n + z')$ has no late job. But this contradicts the minimality of z. $\square$

Thus, assuming that all processing times are integers, the algorithm for $(\alpha \mid \beta \mid L_{max})$ is to solve L_{max} instances of the $(\alpha \mid \beta \mid \sum U_j)$ problem with due dates $(d_1 + z, \ldots, d_n + z)$, z varying from 1 to L_{max}. $\square$

Exercise 1.3. *The above reduction is* not *a polynomial time reduction since the number of iterations is L_{max} which is not necessarily polynomial. How would you make the above reduction polynomial? (Hint: Binary Search).*

Reduction is a very important tool in the study of complexity of any type of problems. As we will see in a few lectures from now, we can classify problems into "easy" and "hard" problems. If a problem A can be reduced to problem B, and problem A is "hard", then we automatically get that problem B is "hard". Conversely, if problem B is "easy", then we automatically get that problem A is "easy".

2 Single Machine Environment

The single machine environment is the easiest machine environment. However, that doesn't mean all scheduling problems in this environment is easy! Moreover, the design of algorithms for the single machine environment gives insight on how to design algorithms for more complicate multiple machine environments. In the next few lectures we will be concentrating on algorithms for the single machine environment, in particular, exact algorithms.

2.1 Minimizing Average Completion Time $(1 \mid\mid \sum C_j), (1 \mid\mid \sum w_j C_j)$

We need to come up with a permutation $\{1, 2, \ldots, n\}$ of the jobs in J such that $\sum_j C_j$ is minimized. Note that in the sum $\sum_j C_j$, the processing time of the job which arrives first is added n times, the second job $n-1$ times and so on. Thus, intuitively, we should process the jobs in increasing order of processing times. This rule is called the *Shortest Processing Time* rule, or simply the SPT rule. The next theorem shows that the intuition above is correct.

Definition 2.1. (SPT): Process the jobs in increasing order of processing time.

Theorem 2.2. *The SPT rule gives an optimal schedule* $(1 \mid\mid \sum C_j)$.

Proof. Suppose the optimal schedule S is not SPT. Then note that there must be a job k and a job ℓ such that $p_k > p_\ell$ and the machine processes k before ℓ. It is not too hard to see that one can assume k and ℓ are consecutive in the schedule. Now consider the schedule S' which is the same as S' except the order of k and ℓ are switched. Note that the completion time of all jobs $j \neq k, \ell$ remains unchanged. Assume that the processing of job k starts at time t in schedule S. Therefore,

$$\sum_j (C_j^S - C_j^{S'}) = [(t + p_k) + (t + p_k + p_\ell)] - [(t + p_\ell) + (t + p_\ell + p_k)] = p_k - p_\ell > 0$$

which is a contradiction since S was the optimal schedule. □

The above argument where we interchanged two jobs which did not satisfy our rule of scheduling to argue about optimality is called the *interchange argument.* A similar argument can be used to show that a generalization of the SPT is optimal for minimizing the weighted completion time.

Definition 2.3. (WSPT): Process the jobs in decreasing order of w_j/p_j.

Theorem 2.4. *The WSPT rule gives an optimal schedule* $(1 \mid\mid \sum w_j C_j)$.

Proof. Suppose the optimal schedule S is not WSPT. Then there must be two jobs k and ℓ such that S processes ℓ right after k but

$$w_k/p_k < w_\ell/p_l$$

Consider the schedule which is the same as S but interchanges the order of k and ℓ. Note that any job other than k and ℓ doesn't change its completion time. Therefore,

$$\sum_j (w_j C_j^S - w_j C_j^{S'}) = [w_k(t+p_k)+w_\ell(t+p_k+p_\ell)] - [w_l(t+p_\ell)+w_k(t+p_\ell+p_k)] = w_\ell p_k - w_k p_\ell > 0$$

which is a contradiction since S is the optimal schedule. $\square$