Scheduling Theory **CO 454, Spring 2009**

Lecture 1: Introduction, Framework and Notation

May 5th, 2009

---

1 Examples of Scheduling Problems

Example 1: Consider a factory `Cart-o-Magik` which produces for its sustenance, carts, which are demanded by various clients. The three major type of carts it has demand for are shopping carts, trolleys for airports, and carts for transportation of goods used while, say moving. Each type of cart goes through a similar series of operations – preparing the metal skeleton, electro-coating the metal, attaching various parts like the base, the wheels, together, final packaging of the carts, etc. However, the machines of `Cart-o-Magik` take different times to do these operations depending on the type of cart being processed.

There are varying demands for the types of carts. Each demand comes with the number of carts required along with a deadline by which these carts are to be delivered. A possible goal of `Cart-o-Magik` is to honour as many demands as possible. How should they the company go about doing this?

Example 2: Consider the central processing unit (CPU) of your laptop/desktop. Different processes are being started by a user at different point of time. For instance, one listens to music on iTunes while changing ones status on Facebook, all the time pretending to read these notes online. As these processes are invoked, the CPU, although fast but having a finite amount of resources, needs to schedule these tasks so that the average time taken for a process to complete is minimized.

Example 3: Consider a busy airport where dozens of planes land and take-off every minute. For the time being consider only the planes landing. Suppose the airport has m-gates in which these landed planes can dock. Also suppose plane i lands at time r_i and moreover takes time p_{ij} to travel from the terminal to the jth gate. Obviously at any point of time a gate can dock at most one plane. How do we decide which plane to send to which gate so that the average time taken by the passengers to get out of the plane is minimized?

All the three examples above are examples of scheduling problems and illustrate the issue of allocating sparse resources over time optimizing an objective at hand. We remark here that different objectives can lead to different solutions and so there is really no one "universally best" schedule. It should also be clear from the above examples that scheduling problems appear in a wide range of fields and a unified approach of studying them is needed. We now describe a very general framework and notation which captures most (but not all) scheduling problems. Our first goal in this course will be to understand this notation thoroughly as henceforth we will not be talking about specific problems but rather problems springing out of the notation.

2 Framework and Notation

Any scheduling problem has associated with it a finite set of tasks or jobs and a finite set of resources or machines. The set of jobs is normally represented as J and is, unless mentioned otherwise, supposed to have n elements, $J = \{1, 2, \ldots, n\}$. The set of machines is normally represented as M and the number of machines, unless mentioned otherwise, is supposed to be m. At any point of time a single machine can process at most one job. Each job $j \in J$ has the following pieces of data associated with it.

Processing Time (p_{ij}) The time taken by machine i to process job j. Many times, the processing time of job j will be independent of the machine, in such cases the processing time will simply be p_j.

Release Time (r_j) Many times a job j is only available for processing after time r_j. This is called the release time of the job. For instance, in Example 3 above, r_j is the time plane j lands.

Due Date (d_j) This is the time the job needs to be processed by. Many a times, completion after the due date is allowed. However a penalty might be incurred for doing so.

Weight (w_j) The weight of a job denotes the relative worth of the job with respect to the other jobs. As we will see later, many a times introducing weights can lead to added complexity in the scheduling problem.

Most scheduling problems can be described by a triplet $(\alpha \mid \beta \mid \gamma)$. The first term α is called the *machine environment* and contains a single entry. This field describes the resources which are available for the completion of various tasks. The second term β denotes the various constraints on the machines and jobs which must be respected by the schedule. The third term γ denotes the objective which is trying to be optimized by the scheduling problem.

Machine Environment (α) The possible machine environments we will study in the course are as follows.

Single Machine Environment ($\alpha = 1$) In this case we have only one machine. Although it might seem a very special case the study of these problems will lead to many techniques useful for more realistic cases.

Identical Parallel Machines ($\alpha = P$) In this case we have m identical machines and any job can run on any machine having the same processing time on each. When the number of machines is constant, say $m = 2$, then the number of machines is appended after the letter P, for instance, $P2$.

Uniform Speed Parallel Machines ($\alpha = Q$) In this case we have m machines and any job can run on any machine. However, each machine i has a speed s_i. The time taken to process job j on machine i is then p_j/s_i.

Unrelated Parallel Machines ($\alpha = R$) In this case we have m machines and any job can run on any machine. However, each job j takes time p_{ij} on machine i. The p_{ij}'s are completely unrelated. For instance, machine i could have $p_{ij} > p_{ij'}$, but machine i' could have $p_{i'j} < p_{i'j'}$.

Open Shop ($\alpha = O$) The following three machine environments fall in the shop scheduling framework. In this framework, each job j consists of m operations and a job is said to be complete if and only if all the operations are completed. Furthermore, each operation takes place on a dedicated machine. Thus each job needs to visit each machine before completion. In the open shop, the jobs can visit the m machines in any order.

Flow Shop ($\alpha = F$) In the flow shop, each job needs to visit the machines in the same fixed order, which is assumed to be $\{1, 2, \ldots, m\}$. One can think of the process as a job visiting a machine and on completion entering the queue of the next machine. A machine on completion of a job chooses to process any job in its queue. In many applications, the machines *also* needs to process the jobs in the order the jobs enter the queue. Such schedules are called *FIFO schedules* and the flow shop is referred to as the *permutation* flow shop and is included in the β-field (see below).

Job Shop ($\alpha = J$) In a job shop, each job comes with a specified order in which it needs to be processed by the m machines. This is the most general of the scheduling problems we will encounter in the course.

Side Constraints (β) The side constraints capture the various restrictions on the scheduling problem. We note that there could be more than one side constraints. The various side constraints we will encounter in this course are

Release Dates ($\beta = r_j$) Unless specified, we assume that all jobs are available from the beginning.

Setup Times ($\beta = s_{jk}(i)$) Many a times a machine $i$needs to spend a set-up time after the completion of job j and beginning the job k. For example, probably the machine needs to be cooled down after job j before starting job k. Unless mentioned these set-up times are assumed to be 0.

Precedence Constraints ($\beta = prec$) Many times a job j cannot be processed until a job k is finished. Such constraints are called precedence constraints. We assume that these constraints are not cyclical, that is, we do not have a situation like job j precedes k, job k precedes l and job l precedes j. One represents the precedence constraint via a *directed acyclic graph* (DAG) where the nodes are the various jobs, and an arc from j to k implies j precedes k in the constraint.

Preemption ($\beta = prmp$) A job can be *preempted* if it is not necessary that the processing be completed once it has started. If jobs can be preempted then a schedule can complete a fraction f of the job in one machine and a fraction $(1 - f)$ in some other machine. The total time to complete is in the proportion of the fractions. By default, we assume that preemption is not allowed. If it is, then the constraint is present in the β field.

Objective (γ) The objective function decides how the scheduling algorithm is designed. However, there is a large list of possible objective functions depending on the application. However, there are still a few which are fundamental and which we will be concerned with. First we start with the following definition. Given a job j, the *completion time* in a schedule S, is the time when the job is processed. This is denoted by C_j^S. The superscript is dropped when the context is clear. When jobs have due dates, the *lateness* of a job denotes the difference between the completion time and the due date. This is denoted as L_j. Thus

$$L_j = C_j - d_j$$

Note that if L_j is negative, the job is not late. An associated measure is called the *tardiness* of a job which the maximum of the lateness and 0. That is, $T_j = \max(L_j, 0)$. Finally, we use U_j to capture if a job j is finished before or after the deadline. If a job j has $C_j > d_j$, we say that the job is *tardy* and let $U_j = 1$. We have $U_j = 0$ otherwise. The various objective functions we will study are as follows.

Makespan ($\gamma = C_{max}$) Find a schedule which minimizes the maximum completion time, that is, $C_{max} := \max_{j \in J} C_j$.

Total Weighted Completion Time ($\gamma = \sum w_j C_j$). Find a schedule which minimizes the weighted average time taken by a job to complete. When all weights are 1, we simply use $\sum C_j$ in the γ-field. This measure is also called the *flow time* or the *weighted flow time.*

Maximum Lateness ($\gamma = L_{max}$) Find a schedule which minimizes the maximum lateness of a job, that is, minimize $L_{max} := \max_{j \in J} L_j$.

Weighted number of tardy jobs ($\gamma = \sum w_j U_j$) Find a schedule which minimizes the weighted number of tardy jobs. When all weights are 1, we simply use $\sum U_j$ in the γ-field.

Observe that all the above objective functions are *nondecreasing* in $C_1, \ldots, C_n$. That is, if we take two schedules S, S', and the completion times are such that $C_j^S \le C_j^{S'}$ for all j, then the objective value of S is smaller than that in S'. Such objective functions or performance measures are called *regular* performance measures.

All the above performance measures are regular. In class we looked at the performance measure $\sum U_j$.

Claim 2.1. *The performance measure $\sum U_j$ is regular.*

Proof. Consider two schedules S and S' such that $C_j^S \le C_j^{S'}$ for all jobs $j \in J$. Therefore, if for a job j, $C_j^S > d_j$, then $C_j^{S'} > d_j$ as well. This implies if $U_j^S = 1$ for job j, then $U_j^{S'} = 1$ as well. So, $\sum_j U_j^S \le \sum_j U_j^{S'}$ implying $\sum U_j$ is regular. □

Exercise 2.2. *Prove that all the above performance measures are regular.*

Not all performance measures are regular. For instance, there are scheduling problems where each job comes j with a time window $[a_j, b_j]$ and the job needs to be processed in that particular window. Let $X_j = 0$ if the job is processed in that time window, and $X_j = 1$ otherwise. Consider the performance measure $\sum X_j$. Is this regular?