Scheduling (single machine)

Logistics

Giovanni Righini

Università degli Studi di Milano

Scheduling problems

Scheduling problems are typically short-term decisions, arising at a tactial or operational level, when operations are considered in full detail and time is taken into account: for each lot to be produced on a machine a starting time must be decided, according to a suitable job sequence: the schedule.

The schedule includes the sequence but possibly additional information, such as idle times, preemptions and so on.

Scheduling problems are typically very difficult to solve to proven optimality.

They involve assignment decisions to assign jobs to machines (binary variables).

They involve sequencing decisions, that can be represented in several ways.

Scheduling problems

In general, we consider a set N of n jobs (indexed by j) and a set M of m machines (indexed by i).
Each job $j \in N$ can be characterized by

- a processing time p_j
- a release date r_j
- a due date d_j
- a deadline D_j
- a weight w_j
- a delay cost c_j

Some of these data may have two indices if their values depend also on the machine.
An available time horizon T is often given.
The objective is to schedule all jobs, minimizing a function of their completion times.

Classification

In 1979 Graham et al. introduced a three fields notation to classify scheduling problems

$$\alpha|\beta|\gamma$$

where

- α describes the machines
- β describes the jobs
- γ describes the objective.

Classification: α

Some possible values for α are:

- 1: single machine
- P (identical): the processing time of each job does not depend on the machine;
- Q (uniform): the machines have different speed s_i and the processing time of job j on machine i is $p_{ij} = p_j / s_i$;
- R (unrelated): the values of p_{ij} are in general different and unrelated;
- F (flow shop): each job must be processed by each machine according to a fixed machine sequence which is the same for all jobs;
- J (job shop): as in flow shop but the machine sequence can be different for each job;
- FJ (flexible job shop): machines are replaced by work centers equipped with parallel identical machines;
- O (open shop): all jobs must be processed on all machines but without routing restrictions.

Classification: β

Some possible values for β are:

- r_j: with release dates;
- *prmp*: with preemption (and resume);
- with precedence constraints, represented by a digraph G with a node for each job:
  - *prec*: G is generic;
  - *chain*: G is a set of paths;
  - *tree*: G is an oriented tree;
- *brkdwn* (breakdown): with fixed unavailability periods for the machines;
- s: with set-up times (dependent or independent of the machine);
- $p_j = 1$: unitary processing times.
- *nwt*: with *no-wait* requirement: for each job in a flow shop, as soon as a machine finishes processing the job, another one must start processing it.

Classification: KPIs

Some typical KPIs in scheduling:

- C_j: completion time of job j;
- $L_j = C_j - d_j$: lateness of job j (it can be negative);
- $T_j = \max\{L_j, 0\}$: tardiness of job j (non-negative);
- $U_j = 1$ if $C_j > d_j$ and 0 otherwise.

An objective function is *regular* if it is non-decreasing with the completion times.

Classification: γ

Some possible values for γ are:

- C_{max}: minimize the completion time of the last job (*makespan*);
- L_{max}: minimize the maximum lateness;
- $\sum C_j$: minimize the total completion time;
- $\sum w_j C_j$: minimize the total weighted completion time;
- $\sum (w_j) T_j$: minimize the total (weighted) tardiness;
- $\sum (w_j) U_j$: minimize the total (weighted) number of tardy jobs.

All these objective functions are regular.

Non-delay schedules

A feasible schedule is *non-delay* if no machine is kept idle, when a job is waiting for being processed.

A feasible non-preemptive schedule is *active* if there is no other schedule obtained from different sequences such that at least one operation finishes earlier and no operation finishes later.

A feasible non-preemptive schedule is *semi-active* if there is no other schedule obtained from the same sequence such that at least one operation finishes earlier.

Single-machine scheduling

Single-machine scheduling problems arise as sub-problems when multi-machine scheduling problems are solved via heuristics or decomposition methods (e.g. Dantizg-Wolfe decomposition).

For some single-machine scheduling problems, the optimal solution can be computed by sorting the jobs according to specific ordering criteria.

Some examples are:

- $1||\sum_j w_j C_j$
- $1|r_j, prmp|\sum_j C_j$
- $1||L_{max}$
- $1||\sum_j U_j$

Several single-machine scheduling problems can be solved in polynomial or pseudo-polynomial time by dynamic programming.

General properties

We consider regular objective functions.

In general, optimal schedules are non-preemptive and non-delay.

If release dates are considered, then the optimal schedule may be preemptive.

If release dates are considered and preemption is not allowed, then the optimal schedule may contain unforced idleness (i.e. it is not non-delay).

Total weighted completion time

$1||\sum_j C_j$ (unweighted case).

SPT (Shortest Processing Time) rule (Smith's rule): sort the jobs by non-decreasing processing time.

Proof: by exchange.
Complexity: $O(n \log n)$.

$1||\sum_j w_j C_j$ (weighted case).

WSPT (Weighted Shortest Processing Time) rule (Smith's rule): sort the jobs by non-increasing values of the ratio w_j / p_j.

Proof: by exchange.
Complexity: $O(n \log n)$.

Proof

By contradiction, assume the schedule is not WSPT. Then there exist two consecutive jobs i and j with i preceding j, such that $\frac{w_i}{p_i} < \frac{w_j}{p_j}$, which implies

$$w_i p_j < w_j p_i.$$

We prove that swapping i and j improves the objective function. We indicate with a prime the values after the swap.

$$C_i' = C_i + p_j$$
$$C_j' = C_j - p_i$$

The completion times of all jobs before i and after j remain unchanged.

Then $z' - z = w_i(C_i' - C_i) + w_j(C_j' - C_j) = w_i p_j - w_j p_i < 0$.

Precedence constraints: chains

Consider two chains of jobs: Chain I consists of jobs $1, \ldots, k$ and Chain II consists of jobs $k + 1, \ldots, n$.

The chains impose the precedences

$$1 \to 2 \to \ldots \to k$$

$$k + 1 \to k + 2 \to \ldots \to n.$$

Assume that the two chains have to be processed with no interruption.

Lemma 1. If

$$\frac{\sum_{j=1}^{k} w_j}{\sum_{j=1}^{k} p_j} > (<) \frac{\sum_{j=k+1}^{n} w_j}{\sum_{j=k+1}^{n} p_j}$$

then it is optimal to process Chain I (Chain II) first.

Proof. Trivial: compare the two objective function values.

The ρ factor

The ρ factor of a chain $1, \ldots, k$ is

$$\rho(1, \ldots, k) = \max_{1 \leq l \leq k} \left\{ \frac{\sum_{j=1}^{l} w_j}{\sum_{j=1}^{l} p_j} \right\}.$$

Let l^* be the job that determines the ρ factor of the chain.

Assume now that chains can be interrupted.

Lemma 2. There exists an optimal sequence that processes jobs $1, \ldots, l^*$ with no interruption.

Proof

Proof. By contradiction, assume a job v from another chain is processed between 1 and l^*:

$$S = 1, \ldots, u, v, u+1, \ldots, l^*.$$

We prove that either $S' = v, 1, \ldots, l^*$ or $S'' = 1, \ldots, l^*, v$ is better than S. From Lemma 1,

$$z(S) < z(S') \Leftrightarrow \frac{w_v}{p_v} < \frac{\sum_{j=1}^{u} w_j}{\sum_{j=1}^{u} p_j}$$

$$z(S) < z(S'') \Leftrightarrow \frac{w_v}{p_v} > \frac{\sum_{j=u+1}^{l^*} w_j}{\sum_{j=u+1}^{l^*} p_j}.$$

By definition of l^*,

$$\frac{\sum_{j=u+1}^{l^*} w_j}{\sum_{j=u+1}^{l^*} p_j} > \frac{\sum_{j=1}^{u} w_j}{\sum_{j=1}^{u} p_j}.$$

Therefore $z(S) < z(S')$ and $z(S) < z(S'')$ cannot be both true.

Total weighted completion time with chains: $1|chain|\sum w_j C_j$

Algorithm. Whenever the machine is freed, select the remaining chain with the largest ρ factor and process it with no interruption up to the job that determines its ρ factor.

Polynomial time algorithms have been devised for other variants of $1|prec|\sum w_j C_j$.

However, the general case with arbitrary precedence constraints is strongly *NP*-hard.

Also $1|r_j, prmp|\sum w_j C_j$ is strongly *NP*-hard, while $1|r_j, prmp|\sum C_j$ is easy.

Preemption and unit weights: $1|r_j, prmp|\sum C_j$

$1|r_j, prmp|\sum_j C_j$.

This problem has unit weights, release dates and preemption.

SRPT (Shortest Remaining Processing Time) rule: sort the jobs by non-decreasing residual processing time.

Proof: by exchange.
Complexity: $O(n \log n)$.

The non-preemptive version $1|r_j|\sum C_j$ is strongly NP-hard.

Single-machine scheduling with due dates

$1|prec|h_{max}$, where

$$h_{max} = \max_{i=1,\ldots,n} \{h_i(C_i)\}$$

and $h_i()$ are non-decreasing cost functions.

This general problem can be solved to optimality with a backward greedy algorithm, even in the case with arbitrary precedence constraints.

We denote by

- J, the set of jobs already scheduled,
- I, the set of jobs to be scheduled,
- $I' \subseteq I$, the subset of schedulable jobs.

Greedy algorithm

The completion of the last job occurs at $C_{max} = \sum_{j=1}^{n} p_j$.

Algorithm.

- Initialize: $J := \emptyset$, $I := \{1, \ldots, n\}$ and $t := C_{max}$.
- Initialize I' as the subset of jobs with no successors.
- Loop: Select $k \in I'$ such that

$$k = \text{argmin}_{i \in I'} \{h_i(t)\}.$$

- Update: delete k from I, insert k in J, $t := t - p_k$.
- Update I' as the subset of jobs with no successors in I.
- If $I = \emptyset$, then stop; otherwise, repeat the loop.

Correctness and complexity

Correctness. Assume job $j \neq k$ is selected at a given iteration, instead of job $k = \text{argmin}_{i \in I'}\{h_i(t)\}$.

Then in the final sequence S, job k occurs before job j and job j ends at time t.

Consider S' obtained from S by moving k immediately after j.

The only job whose completion time in S' is larger than in S is job k and it ends at time t in S'.

But $k = \text{argmin}_{i \in I'}\{h_i(t)\}$ implies that $h_k(t) \leq h_j(t)$.

Hence S' is not worse than S.

Complexity $O(n^2)$. There are n iterations. The selection of k requires $O(n)$. The update of I' requires $O(n)$ (scan all predecessors of k and decrease by 1 their counter of scheduled successors). The initialization may also take $O(n^2)$ to count the number of successors of each job.

Maximum lateness

$1||L_{max}$.

This is the special case where $h_j(C_j) = C_j - d_j \;\; \forall j = 1, \ldots, n$.

EDD (Earliest Due Date) rule (Jackson's rule): sort the jobs by non-decreasing due dates.

Proof: by exchange.
Complexity: $O(n \log n)$.

Proof

By contradiction, assume the schedule is not EDD. Then there exist two jobs i and j with i preceding j, such that $d_i > d_j$. We prove that swapping i and j does not worsen the objective function.
We indicate with a prime the values after the swap.

The following relations hold:

$$L_j = C_j - d_j$$
$$L_i' = C_i' - d_i$$
$$C_i' = C_j$$

Then $L_i' = c_i' - d_i = c_j - d_i \leq c_j - d_j = L_j$.
Since i is the only job whose completion time is increased by the swap, then no lateness value L_k' is larger than L_k for any job $k = 1, \ldots, n$.

Generalizations

$1|r_j|L_{max}$.

The problem is *NP*-hard. Its optimal solution is not necessarily a no-delay schedule.

Branch-and-bound algorithm.
Branching. Fix the next job in all possible ways, discarding *j* when $r_j > \min_{k \notin S}\{\max\{t, r_k\} + p_k\}$, where *S* is the partial schedule and $t = \sum_{j \in S} p_j$ (another job *k* can be completed before r_j).

Lower bound. Schedule the remaining jobs according to a preemptive EDD rule, which is optimal for $1|r_j, prmp|L_{max}$. If the optimal solution of $1|r_j, prmp|L_{max}$ is non preemptive at a node of the BnB tree, then it provides an upper bound for the whole problem.

$1|r_j, prec|L_{max}$ can be solved in a similar way, yielding a smaller BnB tree thanks to the precedence constraints.

Number of tardy jobs

$1||\sum_j U_j$.

Moore algorithm.

- sort the jobs by non-decreasing due dates (EDD)
- start with an empty schedule
- for each job j in the EDD ordered list
    - append j to the schedule
    - if j is late, then select the longest job k in the schedule and remove it
- add all the removed jobs at the end of the schedule (in any order).

Complexity. It can be implemented to run in $O(n \log n)$ using a binary heap to represent the scheduled jobs, so that insertions and deletions take $O(\log n)$ (there are $O(n)$ insertions and $O(n)$ deletions).

Correctness. The proof is omitted (see Pinedo, page 49).

Weighted number of tardy jobs

$1||\sum_j w_j U_j$.

The weighted version of the problem is NP-hard.

When all due dates are equal, then the problem is a Knapsack Problem, where the due date is the capacity of the knapsack.

Total tardiness: lemma 1

$1||\sum_j T_j$.

This problem is weakly *NP*-hard (1990).

Lemma 1. If $p_j \leq p_k$ and $d_j \leq d_k$, then there is an optimal sequence in which job j is scheduled before job k.

Consider an instance with a due dates vector d; consider a job k. Let C'_k be the maximum completion time of job k in an optimal solution S'.

Consider a modified instance where d_k is replaced by $\overline{d}_k = \max\{d_k, C'_k\}$. Let C'' be the completion times vector in an optimal solution S'' of the modified instance.

Total tardiness: lemma 2

Lemma 2. Any sequence that is optimal for the modified instance is also optimal for the original instance.

If $C'_k \leq d_k$, then the two instances are identical and the proof is trivial.

Let z' and z'' indicate the total tardiness in the original and the modified instance, resp..

$$z'(S') = z''(S') + A_k$$
$$z'(S'') = z''(S'') + B_k$$

If $C'_k > d_k$, then

$$A_k = C'_k - d_k$$
$$B_k = (C''_k - d_k) - (C''_k - \overline{d}_k)\}$$

Then $B_k = \overline{d}_k - d_k = \max\{0, C'_k - d_k\} \leq C'_k - d_k = A_k$.
Since $z''(S'') \leq z''(S')$ and $B_k \leq A_k$, then $z'(S'') \leq z'(S')$ (q.e.d.).

Total tardiness: assumptions

Assume w.l.o.g. that all processing times are different.

Renumber the jobs according to the EDD criterion.

Let $p_k = \max_j \{p_j\}$.

From Lemma 1, there is an optimal sequence in which all jobs $1, \ldots, k-1$ are scheduled, in some order, before job k. The other $n-k$ jobs may be scheduled before or after job k.

Total tardiness: lemma 3

Lemma 3. There is an integer $0 \leq \delta \leq n - k$ such that there is an optimal sequence in which job k is preceded by all jobs j with $j \leq k + \delta$ and it is followed by all jobs j with $j > k + \delta$.

Proof. Using the same notation of the previous Lemma, let S'' be a sequence which is optimal when d_k is replaced by $\overline{d}_k = \max\{C'_k, d_k\}$ and complies with Lemma 1; let C''_k be the completion time of job k in S''. By Lemma 2, S'' is also optimal with respect to the original due dates. Therefore $C''_k \leq \overline{d}_k$.

If any job j precedes job k in S'' and $d_j > \overline{d}$, then it can be reinserted after job k still being on time, without increasing the total tardiness.

By Lemma 1, if S'' is optimal, then all jobs j with $d_j < \overline{d}$ must precede job k.

Then δ can be chosen as the largest integer such that

$$d_{k+\delta} \leq \overline{d}_k.$$

Total tardiness: dynamic programming

Owing to Lemma 3, we are guaranteed that

- for any subset of jobs $j, \dots, l$,
- for each starting time t of their schedule,

an optimal sequence is obtained for some $0 \leq \delta \leq l - k$ by the concatenation of three subsequences:

- jobs $j, \dots, k-1, k+1, \dots, k+\delta$, in some order;
- job k (the job with maximum processing time in the subset $\{j, \dots, l\}$);
- jobs $k+\delta, \dots, l$ in some order.

The completion time of job k is $C_k(\delta) = \sum_{i \leq k+\delta} p_i$.

The first and the third subsequences must be optimized and this originates the recursion.

Total tardiness: dynamic programming

To execute the D.P. algorithm bottom-up, one must consider subsets with a special structure: they are subsets $\{j, j+1, \dots, l-1, l\}$ with processing times smaller than p_k for some k. We indicate these sets with $J(j,l,k)$. Let $z(J(j,l,k),t)$ be the minimum total tardiness for $J(j,l,k)$ starting at time t.

D.P. algorithm.
Initialization (recursion base).

$$z(\emptyset, 0) = 0 \qquad z(\{j\}, t) = \max\{0, t + p_j - d_j\}.$$

Extension rule (recursive step).

$$z(J(j,l,k),t) = \min_{\delta}\{z(J(j,k'+\delta,k'),t) + \max\{0, C_{k'}(\delta) - d_{k'}\} + \\ + z(J(k'+\delta+1,l,k'), C_{k'}(\delta))\},$$

where k' is the maximum processing time job in the subset $J(j,l,k)$.

Optimal value: $z(\{1,\dots,n\},0)$.

Complexity: $O(n^4 \sum_j p_j)$, which is pseudo-polynomial.

Models (ILP)

Time-indexed formulations. They use binary variables x_{jt} to indicate whether the execution of job j starts at time t or not, with assignment constraints

$$\sum_{t=0}^{T} x_{jt} = 1 \;\; \forall j \in J$$

and no-overlap constraints

$$\sum_{j=1}^{n} \sum_{s=\max\{t-p_j,0\}}^{t-1} x_{js} = 1 \quad \forall t = 0, \ldots, T$$

. The completion time of each job is given by

$$C_j = \sum_{t=0}^{T} t * x_{jt} + p_j .$$

The constraints and the objective are very easy to formulate, but the number of variables can be huge.

Models (ILP)

Sequencing variables. Binary variables x_{jk} indicate whether job j precedes job k or not, for each pair $j \neq k$.

The completion time of each job is given by

$$C_j = \sum_{k \neq j} p_k x_{kj} + p_j .$$

The following constraints are imposed:

$$\begin{array}{ll}
x_{jk} + x_{kj} = 1 & \forall j, k \in J, j \neq k \\
x_{jk} + x_{kl} + x_{lj} \leq 2 & \forall j, k, l \in J, j \neq k, k \neq l, l \neq j \\
x_{jk} \in \{0, 1\} & \forall j, k \in J \\
x_{jj} = 0 & \forall j \in J .
\end{array}$$

Models (ILP)

Disjunctive programming. Disjunctive constraints are inserted:

$$x_k + p_k \leq x_j \quad \vee \quad x_j + p_j \leq x_k$$

for each pair of jobs, where x_j indicates the start time of job j.

The problem is solved without the disjunctive constraints; the constraints are then checked; if no constraint is violated then the solution is optimal; otherwise binary branching is done, generating a branch-and-bound tree.