CS 524: Introduction to Optimization
Lecture 24 : Scheduling problems

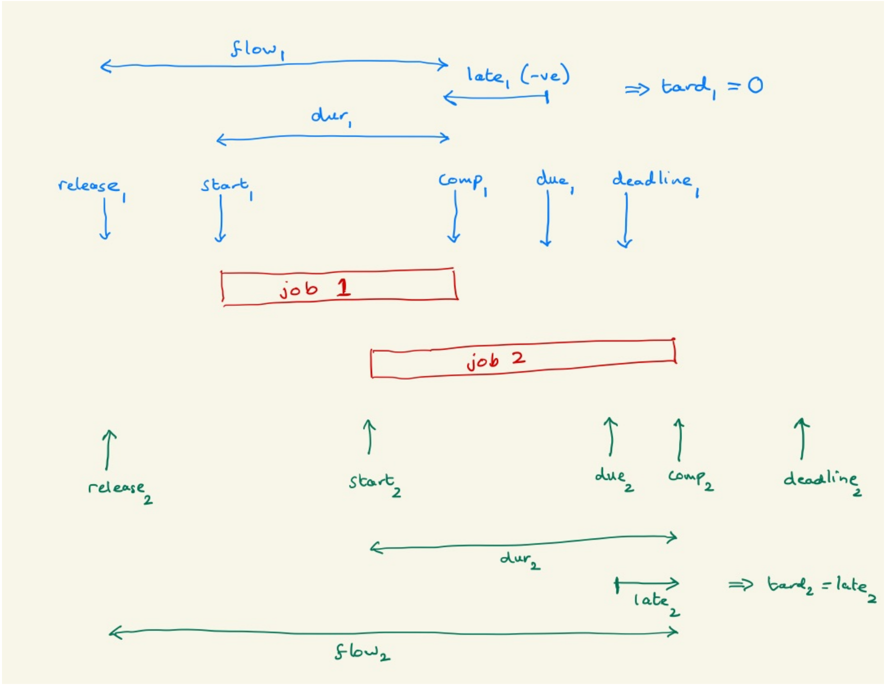
Michael Ferris

Computer Sciences Department
University of Wisconsin-Madison

October 30, 2023

Ordering and scheduling

- Ordering, in this context, refers to the arrangement of events in a certain domain, subject to some constraints. An example from compilers would be the ordering of instructions inside a software pipelined loop for maximum throughput.
- The constraint logic programming constructs are particularly useful in ordering problems since they can easily encode the notions of an ordering.
- We outline here two of the main ideas applicable when a pure MIP approach is used.
- Scheduling or sequencing problems are typically not well solved by MIP formulations. The MIP formulations are limited in size of problems that can be solved in reasonable time frames.
- Much research is currently underway to improve the size of problems that are practically tractable.


flow$_1$
late$_1$ (-ve)
$\Rightarrow$ tard$_1$ = 0
dur$_1$
release$_1$
start$_1$
comp$_1$
due$_1$
deadline$_1$
job 1
job 2
release$_2$
start$_2$
due$_2$
comp$_2$
deadline$_2$
dur$_2$
late$_2$
$\Rightarrow$ tard$_2$ = late$_2$
flow$_2$

Notation and data

- Due date: this is a target time, typically used in an objective function. Try to get close to due date. Compare this with the notion of a deadline, which is typically a hard constraint.
- Release date: the time that a job is released (from other prior processing, for example) and becomes available for processing.
- Duration: how long this task actually takes.
- Completion time:

$$\mathrm{comp}_j = \mathrm{start}_j + \mathrm{dur}_j$$

- Maximum completion time is called makespan.

$$\mathrm{makespan} = \max_j \{\mathrm{start}_j + \mathrm{dur}_j\}$$

- Could also minimize mean/average completion time.

- Flow time:

$$\text{flow}_j = \text{start}_j + \text{dur}_j - \text{release}_j$$

This is time in process. Can look at max or mean flow time.

- Lateness:

$$\text{late}_j = \text{start}_j + \text{dur}_j - \text{due}_j$$

How late the job is - note that the job can be early, in which case lateness is negative. Objective may be max or mean lateness.

- Tardiness:

$$\text{tard}_j = \max\{0, \text{start}_j + \text{dur}_j - \text{due}_j\} = \max\{0, \text{latee}_j\}$$

How tardy the job is completed. Can optimize max or mean tardiness.

sequence2.gms

One type of formulation involves ordering. To represent the start and stop times of a particular task, we use the variables *start*$_i$ and *comp*$_i$. The model implements the condition that either item j finishes before item i starts or the converse using binary order_{ij}:

$$\mathrm{comp}_i \leq \mathrm{start}_j + M(1 - \mathrm{order}_{ij}) \text{ and } \mathrm{comp}_j \leq \mathrm{start}_i + M\mathrm{order}_{ij}$$

Such either-or constraints are termed disjunctions.

`sequence1.gms`

Another formulation of ordering uses binary variables `rank` with the definition:

$$\mathrm{rank}_{ik} = 1 \text{ if item } i \text{ has position } k.$$

The model then includes assignment type constraints that indicate each position k contains one item, and each item i has exactly one rank:

$$\sum_i \mathrm{rank}_{ik} = 1, \forall k \text{ and } \sum_k \mathrm{rank}_{ik} = 1, \forall i.$$

It is then (fairly) straightforward to generate expressions for entities such as the completion time of the item in position k, for example, and thereby generate expressions for waiting time and other objectives.

$$\mathsf{comp}_k \geq \mathsf{comp}_{k-1} + \sum_j \mathsf{dur}_j \mathrm{rank}_{jk}$$

`sequence4.gms`

Alternatively, introduce additional "violation" variables that measure how much the pair (i, j) violates the condition that i finishes before j starts (and the converse):

$$\mathrm{comp}_i \leq \mathrm{start}_j + \mathrm{violation}_{ij} \text{ and } \mathrm{comp}_j \leq \mathrm{start}_i + \mathrm{violation}_{ji}$$

We then add a condition that only one of these two variables $\mathrm{violation}_{ij}$ and $\mathrm{violation}_{ji}$ can be positive, that is we force each such pair of variables to be in an `SOS1` set of size 2 of the form $\{\mathrm{violation}_{ij}, \mathrm{violation}_{ji}\}$. Some solvers (e.g. CPLEX) implement the notion of indicator constraints which provide an alternative way to formulate disjunctions (`sequence3.gms`).

Computations (only demo 1 and 2)

sequence1	rank approach, seems best
sequence2	order approach, either/or and bigM
sequence3	order approach, indicator constraints
sequence4	order approach, indicator via sos1
sequence5	linear ordering approach, Keha (release times = 0 assumed)

Dyeing plant `dyeing.gms`

A fabric dyeing plant has 3 dyeing baths. Each batch of fabric must be dyed in each bath in the order: first, second, third bath. The plant must colour five batches of fabric of different sizes. Dyeing batch i in bath j takes a time s_{ij} expressed in hours in the time matrix below.

```
table time(fabric,baths) 'in hours'
   b1  b2  b3
f1 3   1   2
f2 2   1.5 1
f3 3   1.2 1.3
f4 2   2   2
f5 2.1 2   3  ;
```

Schedule the dyeing operations in the baths so that the ending time of the last batch is minimized.

- For each fabric f, start in next bath after completed in previous bath
- Fabrics f and g cannot be in same bath at same time.
- Flowshop model: machines are baths, jobs are (dyeing) fabrics

Special case: Flow-shop Scheduling

- A flow-shop is a shop design in which machines are arranged in a line.
- There are *n* machines and *m* jobs. Each job contains exactly *n* operations with known execution time. No machine can perform more than one operation simultaneously.
- Operations within one job must be performed in the specified order. The first operation gets executed on the first machine, then (as the first operation is finished) the second operation on the second machine, and so on until the *n*th operation.
- Jobs can be executed in any order. The problem is to determine the optimal arrangement of jobs, i.e. the one with the shortest possible total makespan
- A general flow-shop is somewhat different, in that a job may skip a particular machine. For instance, although every job must proceed from 1 to N, some jobs may go from machine 1 to, say, machine 3 and then machine 4.
- Permutation flow shop scheduling requires the job order to be exactly the same for each machine.

Job-shop Scheduling: `jobshop.gms`

- There are m machines and n jobs to be processed.
- Each job i requires a set O_i of operations in a specific order (maybe given by precedence graph), but the order can be different for each job.
- Some variants have one operation for each machine.
- Each machine can only process one operation at a time, and only one operation in a job can be processed at a given time.
- Various performance evaluation criteria, including makespan.
- Example: Car repair each operator (mechanic) evaluates plus schedules, gets material, etc.