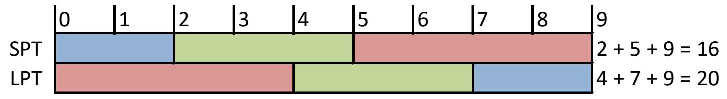
Production control

- Single machine models
  - Prioritizing
  - Optimizing models
  - Optimal algorithms
  - Heuristics
  - Models with set-ups
- Flow shops and lines
- Parallel machines
- Job shops
- FM-systems
- Lean methods
  - CONWIP
  - Kanban
  - Bottleneck control

Single machine scheduling

- We assume that we have a set of jobs available for scheduling
- If there are no constraints related to starting or finishing jobs, a no-wait schedule (no gaps between jobs) is optimal for normal criteria
- Makespan is not affected by the order of jobs
- We may take that each job in the beginning of the schedule delays all later jobs, therefore
- **Average (and total) flow time (starting from zero) is minimized by Shortest Processing Time (SPT) order**:

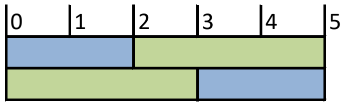


- To minimize WIP, processing times may be weighted by dividing them by the value of jobs (parts)

Total flow time minimization - SPT

Optimality of SPT rule for total flow time is easy to prove as follows:

- Let us examine two consecutive jobs. Independent of the order, makespan, that is finishing time of the last job, is the sum of the processing times
- Therefore only the finishing time of the first job affects the sum of completion times
- The Sum is minimized by scheduling the shorter job first
- In a longer schedule exchanging two consecutive jobs does not affect the timing of other jobs in any way
- By doing exchanges of consecutive jobs so that the shorter job always comes first as long as possible we end up in SPT order



Total flow time minimization – MILP model

- SPT algorithm is a sorting process, which are (can be) polynomial
- This means that the solution time increases in a polynomial relationship to size of the problem (in this case the number of jobs)
- MILP (Mixed Integer Linear Program) optimization model does not utilize this fact but leaves the solution to the solver
- In MILP formulation
  1. Timing of jobs is generated but constrained so that they do not overlap

  or

  2. Order of jobs is generated and timing is constrained so that the next job does not start before the previous job has ended

  or

  3. As a discrete time problem like the alternative 1 above

Total flow time minimization – IP model 1 (”Manne’s formulation”)

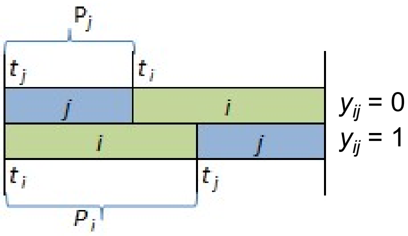
Starting times of jobs are generated, but constrained so that they do not overlap

Constants:

I number of jobs

P_i Duration of job i

M Large number



Decision variables:

t_i starting time of job i

y_{ij} takes value 1, is job i precedes job j, otherwise 0

$$
\begin{array}{ll}
\text{Min } \sum_{\forall i} (t_i + P_i) & \text{st.} \\
t_i \geq 0, & \forall i \\
My_{ij} + (t_i - t_j) \geq P_j, & \forall i \in \{1,..,I-1\}, j \in \{i+1,..,I\} \\
M(1 - y_{ij}) + (t_j - t_i) \geq P_i, & \forall i \in \{1,..,I-1\}, j \in \{i+1,..,I\} \\
y_{ij} \in \{0,1\}, & \forall i,j
\end{array}
$$

Total flow time minimization – IP model 2 (”Wagner’s formulation”)

Order of jobs is generated and timing is constrained so that the next job does not start before the previous job has ended

Now decision variables y_{ij} take value 1, if job i is in position j in the schedule, otherwise 0

$$
\begin{array}{ll}
\text{Min } \sum_{\forall j} \left(t_j + \sum_{\forall i} P_i y_{ij} \right) & \\
t_j \geq 0, & \forall j \\
\sum_i y_{ij} = 1, & \forall j \\
\sum_j y_{ij} = 1, & \forall i \\
t_j + \sum_{\forall i} P_i y_{ij} \leq t_{j+1}, & \forall j \in \{1,..,I-1\} \\
y_{ij} \in \{0,1\}, & \forall i,j
\end{array}
$$

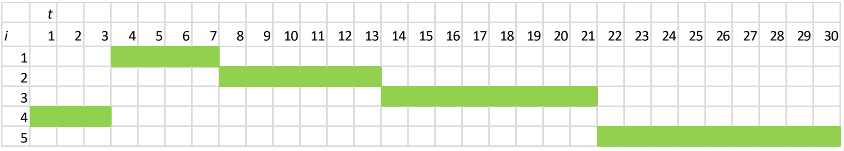
Job, i	P_i	y_{ij} \ j	1	2	3	4	5	
1	4		-0	1	0	0	0	1
2	6		0	0	1	0	0	1
3	8		0	0	0	1	0	1
4	3		1	-0	0	0	0	1
5	9		0	0	-0	0	1	1
			1	1	1	1	1	
		t_j	0	3	7	13	21	
		P_j	3	4	6	8	9	
		c_j	3	7	13	21	30	74

Total flow time – IP

Job, i	P_i	t_i	P_i+t_i	y_{ij}	j: 1	2	3	4	5
1	4	3	7			1	1	0	1
2	6	7	13				1	0	1
3	8	13	21					0	1
4	3	0	3						1
5	9	21	30						
			74						

Model 1

y_{ij} take value 1, if job i precedes j, otherwise 0



Job, i	P_i	y_{ij}	j: 1	2	3	4	5	
1	4		0	1	0	0	0	1
2	6		0	0	1	-0	0	1
3	8		0	0	0	1	0	1
4	3		1	0	0	0	0	1
5	9		0	0	0	0	1	1
			1	1	1	1	1	
		t_j	0	3	7	13	21	
		P_j	3	4	6	8	9	
		c_j	3	7	13	21	30	74

Model 2

y_{ij} take value 1, if job i is in position j in schedule, otherwise 0

Total flow time – discrete time model

- End times x_{it} are generated for jobs and running time is calculated backward and indicated with auxiliary variables y_{it}
- Job overlap is prevented
- Each job ends once

t time

Decision variables:

x_{it} take value 1, if job i ends at time t

y_{it} take value 1, if job i is processed at time t

$$\text{Min} \sum_{\forall i} \sum_{\forall t} t x_{it}$$

$$\sum_{\forall t} x_{it} = 1, \qquad \forall i$$

$$y_{it} = \sum_{u=t}^{t+P_i-1} x_{iu}, \qquad \forall i, t$$

$$\sum_{\forall i} y_{it} \leq 1, \qquad \forall t$$

Timing, y_{it}:

Job, i	Duration, P_i	Time,	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1	4	4				1	1	1	1																							
2	6	6								1	1	1	1	1	1																	
3	8	8														1	1	1	1	1	1	1	1									
4	3	3	1	1	1																											
5	9	9																						1	1	1	1	1	1	1	1	1
		Σ	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

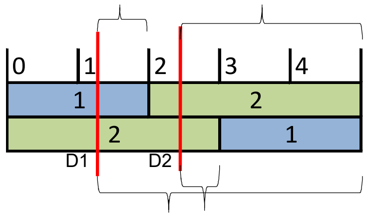
Finishing, x_{it}:

	Aika, t	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	Σ	f_i End
1								1																								1	7
2														1																		1	13
3																						1										1	21
4				1																												1	3
5																															1	1	30
																																	74

Maximum tardiness minimization - EDD

Maximum tardiness is minimized by sorting them in the Earliest Due Date order:

- We examine two consecutive jobs and tardinesses in different alternatives:



- From the figure can be seen, that tardiness is maximized when the job with earlier due date is processed later, because a worse alternative can not exist
- The result can not be worse in the other alternative
- Alternatives, in which no tardiness (or partial) exist are not relevant (or do not change the result)
- By doing exchanges of consecutive jobs as long as possible we end up in EDD order

Maximum tardiness minimization – IP model

IP model based on Manne's model

New constants:

D_i Due date of job i

And decision variable:

f maximum tardiness

Min f	
$t_i + P_i - D_i \le f,$	$\forall i$
$t_i \ge 0,\ f \ge 0,$	$\forall i$
$My_{ij} + (t_i - t_j) \ge P_j,$	$\forall i \in \{1,..,I-1\}, j \in \{i+1,..,I\}$
$M(1 - y_{ij}) + (t_j - t_i) \ge P_i,$	$\forall i \in \{1,..,I-1\}, j \in \{i+1,..,I\}$
$y_{ij} \in \{0,1\},$	$\forall i, j$

Total tardiness minimization

For total tardiness minimization no simple algorithm exists. Many heuristics are based on that

- If the schedule is loose and there are few late jobs, EDD rule probably works well
- If the schedule is tight and most jobs are late, SPT rule probably works well

IP model

New variables f_i express tardiness of job *i*

$$
\begin{array}{ll}
\text{Min } \sum_{\forall i} f_i & \\
t_i + P_i - D_i \le f_i, & \forall i \\
t_i \ge 0,\ f_i \ge 0, & \forall i \\
My_{ij} + (t_i - t_j) \ge P_j, & \forall i \in \{1,..,I-1\}, j \in \{i+1,..,I\} \\
M(1-y_{ij}) + (t_j - t_i) \ge P_i, & \forall i \in \{1,..,I-1\}, j \in \{i+1,..,I\} \\
y_{ij} \in \{0,1\}, & \forall i,j
\end{array}
$$

Minimization of number of tardy jobs

Minimizing number of tardy jobs is necessary when on time delivery is maximized. Efficient algorithms exist for this. For example:

Jobs are first sorted according to due dates, then

1. Find first tardy job *k*
2. Find longest job 1..*k* and remove it and move last in schedule
3. Return to step 2 until there are no more tardy jobs (in the original schedule)

Example:

Job *i*	Duration P_i	Due date D_i
1	1	2
2	5	7
3	3	8
4	9	13
5	7	11

1. {1,2,3,5,4} *k* = 3, longest in 1..3 is job 2
2. {1,3,5,4} {2} *k* = 4, which itself is longest
3. {1,3,5} {2,4}
4. No more late jobs, final schedule: {1,3,5,2,4}

Minimization of number of tardy jobs

IP model for minimization of number of tardy jobs is achieved easily as a modification of Manne's model:

New variables z_i take value 1, if job i is late, otherwise 0

$$
\begin{array}{ll}
\text{Min } \sum_{\forall i} z_i & \\
t_i + p_i - D_i \le M z_i, & \forall i \\
t_i \ge 0, & \forall i \\
M y_{ij} + (t_i - t_j) \ge P_j, & \forall i \in \{1,..,I-1\}, j \in \{i+1,..,I\} \\
M(1 - y_{ij}) + (t_j - t_i) \ge P_i, & \forall i \in \{1,..,I-1\}, j \in \{i+1,..,I\} \\
y_{ij} \in \{0,1\}, & \forall i, j \\
z_i \in \{0,1\}, & \forall i
\end{array}
$$

Single machine scheduling

- The presented optimization models are flexible, because
  - Same models are easy to modify for different objectives
  - Constraints for earliest starting times are easy to add (in this case the optimal rules presented do not apply)
  - Constraints for job precedence are easy to add
- Manne's model is often easier to apply and
  - It is easy to extend to job shops
- Wagner's model directly expresses job order and then
  - Set-up times are easy to model
  - Choice from alternative (parallel) machines is easy to model
- Discrete model has advantages of both continuous time model types, but it is computationally heavy and inaccurate because of the limited resolution

Single machine and set-ups

Total flow time minimization (Wagner's model) with set-ups

- Similar products are grouped to set-up groups
- Set-up time S_k does not realize, if products belonging to same set-up group *k* immediately follow each other
- Set-up time does not depend on the type of previous (different) job

Now variables s_{jk} take value 1, if set-up changes, otherwise 0.
Constants R_{ik} indicate if job *i* belongs to group *k,* in which case R_{ik} = 1

$$
\begin{array}{ll}
\text{Min } \sum_{\forall j} \left(t_j + \sum_{\forall i} P_i y_{ij} + \sum_{\forall k} S_k s_{jk} \right) & \\
t_j \geq 0, & \forall j \\
\sum_i y_{ij} = 1, & \forall j \\
\sum_j y_{ij} = 1, & \forall i \\
t_j + \sum_{\forall i} P_i y_{ij} + \sum_{\forall k} S_k s_{jk} \leq t_{j+1}, & \forall j \in \{1,..,I-1\} \\
Ms_{jk} \geq \sum_{\forall i} R_{ik} y_{ij} - \sum_{\forall i} R_{ik} y_{i,j-1}, \forall j,k & \\
y_{ij} \in \{0,1\}, s_{jk} \in \{0,1\} & \forall i,j,k
\end{array}
$$

s is set to 1 if two consecutive jobs are different

Makespan optimization is a relevant criterion when set-ups are taken into account

Sequence dependent set-up time problem (SDSTP)

- Sequence dependent set-up times are common in manufacturing. Examples:
  - Machining concerning fixtures and tools
  - Sheet metal cutting
  - Painting
  - Injection molding etc.
- Naturally no set-up is necessary here either, if similar jobs can be scheduled consequently. In the next examples the job sequence is not considered from the delivery time perspective
- One popular heuristic is to always choose the job with the smallest set-up time next (nearest neighbor or closest insertion algorithm). The starting job affects the result. Therefore it is useful to try starting from each job
- A MILP model is also possible, although a little complicated

Heuristic solution for SDSTP

- In make-to-order production schedule is made into near future and the last job in sequence is not important
- In make-to-stock production repeating the same cycle may be beneficial

Set-up costs

From: To:

Job	1	2	3	4	5
1	-	18	3	3	6
2	19	-	9	10	5
3	9	18	-	13	20
4	6	6	1	-	2
5	17	1	13	17	-

Closest insertion -algorithm

All jobs once

Sequence	Cost
1 - 3 - 4 - 5 - 2:	3 + 13 + 2 + 1 = 19
1 - 4 - 3 - 2 - 5:	3 + 1 + 18 + 5 = 27
2 - 5 - 3 - 1 - 4:	5 + 13 + 9 + 3 = 30
3 - 1 - 4 - 5 - 2:	9 + 3 + 2 + 1 = **15**
4 - 3 - 1 - 5 - 2:	1 + 9 + 6 + 1 = 17
5 - 2 - 3 - 1 - 4:	1 + 9 + 9 + 3 = 22

Full cycle

Sequence	Cost
1 - 3 - 4 - 5 - 2 - 1:	3 + 13 + 2 + 1 + 19 = 38
1 - 4 - 3 - 2 - 5 - 1:	3 + 1 + 18 + 5 + 17 = 44
2 - 5 - 3 - 1 - 4 - 2:	5 + 13 + 9 + 3 + 6 = 36
3 - 1 - 4 - 5 - 2 - 3:	9 + 3 + 2 + 1 + 9 = **24**
4 - 3 - 1 - 5 - 2 - 4:	1 + 9 + 6 + 1 + 10 = 27
5 - 2 - 3 - 1 - 4 - 5:	1 + 9 + 9 + 3 + 2 = **24**

Optimisation model for sequence-dependent setup time problem

- SDSTP is a variation of the travelling salesman problem, but usually with
  - Asymmetric travelling (setup) times
  - Open sequences
- MILP formulation, closed loop (J = number of jobs, S_{ij} = setup time from i to j, x = binary variable, o = position in sequence, variable)

$$\text{Min} \quad \sum_{i=1}^{J}\sum_{j=1}^{J} x_{ij}S_{ij}$$

so that

$$\sum_{i=1}^{J} x_{ij} = 1 \qquad \forall j \in \{1,2,\dots,J\}$$

$$\sum_{j=1}^{J} x_{ij} = 1, \qquad \forall i \in \{1,2,\dots,J\}$$

$$\underbrace{o_i - o_j + Jx_{ij} \le J - 1}_{\text{You can only go forward in sequence}} \qquad \forall 2 \le i \ne j \le J$$

However, you can jump backward once at i = 1