4 Scheduling

- In this section, we consider so-called "Scheduling problems"
- I.e., if there are altogether *M* machines or resources for each machine, a production sequence of all *N* jobs has to be found as well as the determination of the time tables
- Consequently, we have to decide on
  - The sequence of the respective jobs on each machine
  - and its time table

Outline of the chapter

4.1 Preliminaries

- Production program is given
- Lot sizes are given
- Process sequence of each job is given
- Operating times are given
- No operation can be processed simultaneously on more than one machine
- At each point of time every machine can process at most one job
- At the beginning of the planning horizon all *N* jobs and their data are available (static problem)
- Transports and storage are never bottlenecks
- No maintenance and repair activities
- On each machine setup times are independent of the realized operation sequence

Given and sought

- Given:
  - *MS*: Machine sequence matrix
  - *PT*: Matrix of processing times
- Sought:
  - *JS*: Job sequence matrix
  - *TT*: Timetable planning matrix with:

$t_{m,n} \; (1 \le m \le M; 1 \le n \le N)$

Point of time where the processing of job n at machine m begins [TU]

4.1.1 Mathematical model

Variables:

$t_{m,n}\ (1 \le n \le N; 1 \le m \le M)$: see above

$y_{m,n,k}\ (1 \le m \le M; 1 \le n \le N; 1 \le k \le N)$: Binary variable defining the sequence of jobs, i.e.,

$$y_{m,n,k} = \begin{cases} 1 & \text{if job } n \text{ is processed on machine } m \text{ before job } k \\ 0 & \text{otherwise} \end{cases}$$

Mathematical model

Restrictions:

Machine sequence restrictions (derived from the matrix *MS*):

$$\forall m \in \{1,\dots,M-1\} : \forall n \in \{1,\dots,N\} : t_{[m],n} + p_{[m],n} \le t_{[m+1],n}$$

$[m]$ defines in this connection the index of the machine that executes the m-th operation of job n

Mathematical model

- In case of the job sequence restrictions, the formulation depends on the structure of the found solution
- But, we have to ensure that there is no simultaneous processing of two jobs on any machine, wherefore an arbitrary sequence of those jobs has to be realized

Therefore, there are altogether two possible cases:

First case (n before k):

(1) $t_{m,n} + p_{m,n} \le t_{m,k}$

Second case (k before n):

(2) $t_{m,k} + p_{m,k} \le t_{m,n}$

$\Rightarrow$ Both possibilities have to be considered in the model!

Mathematical model

Restrictions:

Job sequence restrictions (depends on the chosen solution):

$$\forall m \in \{1,\dots,M-1\} : \forall n,k \in \{1,\dots,N\} : t_{m,n} + p_{m,n} \le t_{m,k} + (1 - y_{m,n,k}) \cdot \overline{C}$$
$$\forall m \in \{1,\dots,M-1\} : \forall n,k \in \{1,\dots,N\} : t_{m,k} + p_{m,k} \le t_{m,n} + y_{m,n,k} \cdot \overline{C}$$

$\overline{C}$ defines a big number, which is larger than each definition of timetable variables $t_{m,n}$, e.g., $\overline{C} = \sum_{m=1}^{M} \sum_{n=1}^{N} p_{m,n}$

Mathematical model

Restrictions :

$$\forall m \in \{1,...,M\}: \forall n,k \in \{1,...,N\}(n \neq k): y_{m,n,k} \in \{0,1\}$$

$$\forall m \in \{1,...,M\}: \forall n \in \{1,...,N\}: t_{m,n} \geq 0$$

4.1.2 Objective functions

- The model defined above can be regarded as a general starting point for so-called job-shop scheduling problems
- It abstains from the definition of a particular objective function but can be extended by a specific application-dependent one
- A huge set of different objective functions is proposed in literature. These functions mainly influence the efficiency of applied solution procedures
- In the following, we will give some examples of well-known objectives

Minimization of cycle time

- Here, we consider the duration of producing the total production quantities

$$\text{Minimize } Z_1 = t_{\max} = \max\{C_{[M],n} \mid n \in \{1,...,N\}\}$$

with:

$\forall n \in \{1,...,N\}: C_{[M],n}$: Point of time where the last processing of job n is finalized

Minimization of machine waiting times

- Sum of all machine waiting times throughout all used resources

$$\text{Minimize } Z_2 = \sum_{m=1}^{M} \underbrace{\left(C_{\max} - \sum_{n=1}^{N} p_{m,n}\right)}_{\text{Unused capacity of machine } m}$$

$$= M \cdot C_{\max} - \left(\sum_{m=1}^{M}\sum_{n=1}^{N} p_{m,n}\right)$$

Since $\left(\sum_{m=1}^{M}\sum_{n=1}^{N} p_{m,n}\right)$ and M are constants, Z_1 and Z_2 are equivalent

Minimization of total completion (lead) time

- This objective intends to minimize the total sum of all individual completion or lead times
- Therefore, we compute the sum of dwell times over all jobs

$$\text{Minimize } Z_3 = \sum_{n=1}^{N} C_{[M],n}$$

This objective is equivalent to the minimization of the sum of the waiting times of all jobs

$$\text{Minimize } Z_4 = \sum_{n=1}^{N} \sum_{m=1}^{M} w_{m,n}$$

Minimization of maximum lead time

- Here, we want to minimize the dwell time of the job whose processing takes the longest time among all N jobs
- This is the objective function Z_1

Min. of the sum of due date deviations

- In this case completion time and due date of each job are compared, while the difference is taken as the result and summed up throughout all jobs to be processed
- As a consequence, an early completion gets a bonus while each lateness is punished

$$\text{Minimize } Z_5 = \sum_{n=1}^{N} C_{[M],n} - d_n = \sum_{n=1}^{N} C_{[M],n} - \sum_{n=1}^{N} d_n$$

with:

d_n : Due date of job n

Since $\sum_{n=1}^{N} d_n$ is a constant, this objective is equivalent to Z_3

Minimization of total lateness (or tardiness)

- Here, we want to minimize the total lateness over all N jobs to be produced in the considered production system
- Consequently, there is no longer compensation between early and late deliveries possible

$$\text{Minimize } Z_6 = \sum_{n=1}^{N} \max\left\{C_{[M],n} - d_n, 0\right\}$$

with :

d_n : Due date of job n

Minimization of maximum lateness

- By using this objective, we somehow want to balance the lateness equally among the different jobs in the found solution
- Thus, we try to minimize the maximum lateness of a job in the found solution

$$\text{Minimize } Z_7 = \max\left\{\max\left\{C_{[M],n} - d_n, 0\right\} \mid n \in \{1,\ldots,N\}\right\}$$

with:

d_n : Due date of job n

Minimization of weighted sum of lead times

- Here, each job gets an individual weight rating its dwelling time in the production system
- Altogether, by doing so we receive a combined weighted sum of lead times

$$\text{Minimize } Z_8 = \sum_{n=1}^{N} w_n \cdot C_{[M],n}$$

with:

w_n : Weight for product n

4.1.3 Schedule classes

- In the following, we introduce some basic terms for specific types of schedules
- In scheduling, a distinction is frequently made between
  - Sequence,
  - Schedule and
  - Scheduling policy
- Sequence

  Corresponds to a specific permutation of jobs to be processed on a given machine
- Schedule

  Usually corresponds to an allocation of jobs within a more complicated setting of machines, which could allow for preemptions of jobs by other jobs that are released at later points in time. Comprises time tables
- Scheduling policy

  Often used in stochastic settings; a policy prescribes an appropriate action for any of the states the system may be in. In deterministic cases, usually only sequences or schedules are of importance but can be extended by rule definitions

Non-delay schedules

4.1.3.1 Definition

A feasible schedule is called non-delay if no machine is kept idle when there is an operation available for processing

Active schedules

4.1.3.2 Definition

A feasible schedule is called active if no operation can be completed earlier by starting earlier or changing the process sequence on machines without delaying any other operation

Attributes of active schedules

4.1.3.3 Lemma

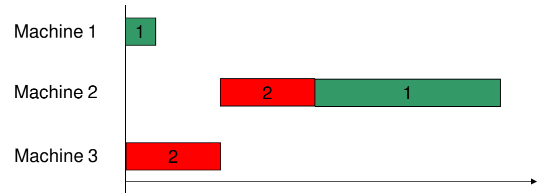
A non-delay schedule is always active

Proof of the lemma

- Let us assume there is a non-delay schedule that is not active
- Then, we know there is a machine where shifting an operation i into an earlier position at point of time t results in an earlier completion without delaying the other operations
- But, if this is true, we know that during the processing of the schedule on machine m there is a constellation at point of time t where the considered machine is idle but can process job i instead
- This is a contradiction to the assumption that the schedule is non-delay

Attributes of active schedules

- Note that the reverse is not necessarily true
- i.e., there are some active schedules that are not non-delay
- Example: Schedule is active but not non-delay



Semi-active schedules

4.1.3.4 Definition

A feasible schedule is called semi-active if no operation can be completed earlier without altering the processing sequence on any of the machines

Consequences

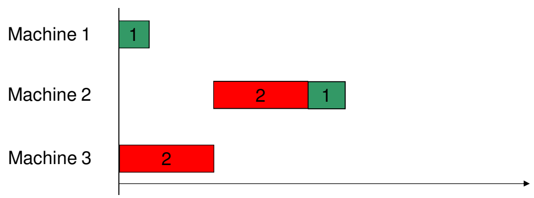
4.1.3.5 Lemma

An active schedule is always semi-active

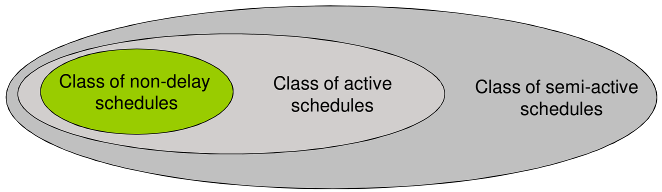
The proof is trivial and immediately results from the definition

Attributes of semi-active schedules

- Note that the reverse is not necessarily true
- i.e., there are some semi-active schedules that are not active
- Example: Schedule is semi-active but not active



Schedule class hierarchy



4.2 Single-Stage Systems

- Now, we consider a single production stage only
- i.e., M=1, wherefore we have only one indexed processing times $p_1,\ldots,p_N$
- Now, the complexity of the models only depends on the considered objective function
- There are some constellations that can be optimally solved in $O(N \log N)$ time using a simple priority rule as well as models that are already *NP*-complete problems. And both happens despite the fact that besides their objective function, both problems are completely the same one-stage problems

Minimization of cycle time

- Trivial problem
- Each solution leads to the same result
- Therefore, an arbitrary solution is already an optimal one

Minimization of weighted sum of lead times

4.2.1 Theorem

The WSPT-rule leads to the optimal solution

Weighted **S**hortest **P**rocessing **T**ime First Rule:

This rule processes all N jobs in the sequence of non-increasing order of the value w_j/p_j

Proof of Theorem

- We will show the claim by contradiction
- Therefore, we assume that there is an optimal sequence of the problem that does not fulfill all the restrictions of the WSPT policy
- Consequently, there are two adjacent jobs, say job j followed by job k, such that

$$w_j/p_j < w_k/p_k$$

Proof of Theorem

- Assume job *j* starts its processing at time *t*
- Let us perform an **interchange of *j* and *k***
- Therefore, the modified schedule starts job *k* now at *t* while **all other jobs remain in their original position**
- Consequently, their weighted objective value is not affected at all and, therefore, remains unchanged
- Call the **old schedule *S*** and the **new modified one *T***

Proof of Theorem

- Under schedule *S*, the total weighted completion of jobs *j* and *k* is rated by:

$$\boxed{(t+p_j)\cdot w_j + (t+p_j+p_k)\cdot w_k}$$

- Under schedule *T*, the total weighted completion of jobs *j* and *k* is rated by:

$$\boxed{(t+p_k)\cdot w_k + (t+p_j+p_k)\cdot w_j}$$

Proof of Theorem IV

$$\underbrace{(t+p_j)\cdot w_j + (t+p_j+p_k)\cdot w_k}_{\text{Objective function value of solution S}} - \underbrace{((t+p_k)\cdot w_k + (t+p_j+p_k)\cdot w_j)}_{\text{Objective function value of solution T}} =$$

$$t\cdot w_j + p_j\cdot w_j + t\cdot w_k + p_j\cdot w_k + p_k\cdot w_k$$

$$-t\cdot w_k - p_k\cdot w_k - t\cdot w_j - p_j\cdot w_j - p_k\cdot w_j =$$

$$p_j\cdot w_k - p_k\cdot w_j$$

It holds: $w_j / p_j < w_k / p_k \Leftrightarrow w_j\cdot p_k < w_k\cdot p_j$

$$\Rightarrow p_j\cdot w_k - p_k\cdot w_j > 0$$

Proof of Theorem V

- Consequently, solution *T* is better and, therefore, the proof is completed

Minimization of total lead time

4.2.2 Corollary

The SPT-rule leads to the optimal solution

Shortest **P**rocessing **T**ime First Rule:

This rule processes all N jobs in the sequence of non-decreasing processing times p_j

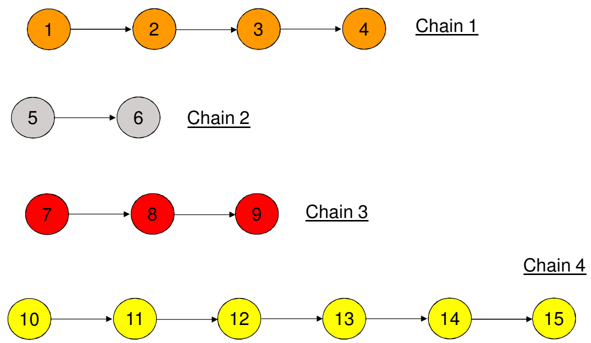
Proof of the Corollary

- To prove the corollary, we may use again Theorem 4.2.1
- To do so, we easily derive that the objective function for minimizing the total lead time Z_3 is a special case of the more general weighted sum of lead time Z_8
- In this case all weights are set to 1
- By applying Theorem 4.2.1, we can derive that we receive the optimal sequence by using the WSPT-policy, i.e., by respecting this special setting, we sort all jobs in non-increasing sequence of the $w_j/p_j=1/p_j$ values
- Consequently, the jobs are sorted in an non-decreasing sequence of the p_j-values as defined by the well-known SPT-rule. This completes the proof of the corollary

Precedence constraints

- How is the result affected by precedence constraints?
- In the following, we introduce additional **precedence constraints** limiting the solution space through the exclusion of some possible solutions. These constraints are very simple and can be described through parallel chains defining which job has to be processed before another one
- This is a situation that frequently occurs during the processing of multi-stage systems
- First, we can process only entire chains. To solve these problems optimally, we can use the following extended Theorem 4.2.3

Parallel precedence chains



Entire chain problem

4.2.3 Theorem

The entire chain problem according to the objective of total weighted lead time minimization can be solved optimally by sorting the chains in non-increasing order of the value:

$\dfrac{\sum_{j=1}^{k} w_j}{\sum_{j=1}^{k} p_j}$ is the value for the chain comprising the jobs 1,2,...,k

Proof of the Theorem

- Again, we show the claim by contradiction
- Therefore, we assume there is an optimal production plan violating the rule definition
- Therefore, there are **two neighbored chains** (1,…,k) and (k+1,…,l) where the defined priority rule is not fulfilled
- Again, we can derive that there is no impact on the weighted lead time of the jobs not belonging to one of the two chains
- Moreover, we derive **schedule *T*** from the **current schedule denoted *S*** through the exchange of the two neighboring chains, i.e., in *T* we process (k+1,…,l) before (1,…,k) is processed
- In what follows, we compute the respective objective values of the both chains for the two possible constellations *S* and *T*

Schedule S – Objective function value

Objective function value under schedule *S*:

$$w_1 \cdot (t+p_1) + w_2 \cdot (t+p_1+p_2) + \ldots + w_k \cdot \left(t + \sum_{j=1}^{k} p_j\right) + w_{k+1} \cdot \left(t + \sum_{j=1}^{k+1} p_j\right) +$$

$$\ldots + w_l \cdot \left(t + \sum_{j=1}^{l} p_j\right) = t \cdot \left(\sum_{j=1}^{l} w_j\right) + \sum_{j=1}^{l} w_j \cdot \left(\sum_{i=1}^{j} p_i\right)$$

Schedule T – Objective function value

Objective function value under schedule T:

$$w_{k+1} \cdot (t+p_{k+1}) + w_{k+2} \cdot (t+p_{k+1}+p_{k+2}) + \ldots + w_l \cdot \left(t + \sum_{j=k+1}^{l} p_j\right)$$

$$+ w_1 \cdot \left(t + \sum_{j=k+1}^{l} p_j + p_1\right) + \ldots + w_k \cdot \left(t + \sum_{j=k+1}^{l} p_j + \sum_{j=1}^{k} p_j\right)$$

$$= t \cdot \left(\sum_{j=1}^{l} w_j\right) + \sum_{j=1}^{l} w_j \cdot \left(\sum_{i=1}^{j} p_i\right) - \sum_{j=k+1}^{l} w_j \cdot \left(\sum_{i=1}^{k} p_i\right) + \sum_{j=1}^{k} w_j \cdot \left(\sum_{i=k+1}^{l} p_i\right)$$

The gap between T and S

$$\sum_{j=1}^{k} w_j \cdot \left(\sum_{i=k+1}^{l} p_i\right) - \sum_{j=k+1}^{l} w_j \cdot \left(\sum_{i=1}^{k} p_i\right)$$

We know: $\dfrac{\sum_{j=1}^{k} w_j}{\sum_{j=1}^{k} p_j} < \dfrac{\sum_{j=k+1}^{l} w_j}{\sum_{j=k+1}^{l} p_j} \Leftrightarrow \sum_{j=1}^{k} w_j \cdot \sum_{j=k+1}^{l} p_j < \sum_{j=k+1}^{l} w_j \cdot \sum_{j=1}^{k} p_j$

Consequently: $\sum_{j=1}^{k} w_j \cdot \left(\sum_{i=k+1}^{l} p_i\right) - \sum_{j=k+1}^{l} w_j \cdot \left(\sum_{i=1}^{k} p_i\right) < 0$

Consequence

- The objective value of schedule *T* is better than the result under schedule *S* and, therefore, the optimality of the rule defined above is shown
- This completes the proof

ρ-factor

4.2.4 Definition

Let us consider a chain of jobs (1,…,*k*). Then, the job *k** out of the chain is called the ***ρ*-factor of the chain (1,…,*k*)** if

$$\frac{\sum_{i=1}^{k^*} w_i}{\sum_{i=1}^{k^*} p_i} = \max\left\{ \frac{\sum_{i=1}^{l} w_i}{\sum_{i=1}^{l} p_i} \,\middle|\, 1 \le l \le k \right\}$$

Allowing preemption

- Assume now that the scheduler has the freedom to process any number of jobs in a chain (while adhering to the precedence constraints) without necessarily having to complete all the jobs in the chain before switching to another chain
- In what follows, we consider again the case of multiple chains
- Moreover, total weighted lead time is assumed to be the objective function
- Then, we may apply the result given in the following Theorem 4.2.5 in order to derive an optimal production plan

Subchain preemption

4.2.5 Lemma

If job l^* is the ρ-factor of the chain (1,…,k), then there exists an optimal sequence that processes jobs 1,…,l^* one after another without interruption by jobs from other chains

Proof of the Theorem

- Again, we prove this claim by contradiction
- Suppose that under the optimal sequence, the processing of the subsequence 1,…,l^* is interrupted by a job, say job v, from another chain that has to be processed simultaneously
- Thus, the optimal sequence contains the subsequence 1,…,u,v,u+1,…,l^*, say subsequence S
- It suffices to show that either with subsequence v,1,…,l^*, say S' or with 1,…,l^*,v, say S'', the total weighted completion time is less than with subsequence S
- We know that the lead time of all other jobs besides 1,…,l^* and v is independent of the chosen subsequence S, S', and S''
- In the following, we therefore assume S'>S as well as S''>S

Case 1: S'>S

- S=(1,…,u,v,u+1,…,l^*); S'=(v,1,…,u,u+1,…,l^*)
- Since S is better than S', we can apply the proof of Theorem 4.2.3 to derive that it holds:

$$\boxed{\frac{w_v}{p_v} < \frac{w_1+\ldots+w_u}{p_1+\ldots+p_u}}$$

- If this is not true, we do not worsen the solution by applying the proof of Theorem 4.2.3 and process v before the job sequence 1,…,u
- It is trivial that we can choose S' instead of S if it holds S'≤S

Case 2: S''>S

- S=(1,…,u,v,u+1,…,l^*); S''=(1,…,u,u+1,…,l^*,v)
- Since S is better than S'', we can again apply the proof of Theorem 4.2.3 to derive that it holds:

$$\boxed{\frac{w_v}{p_v} > \frac{w_{u+1}+\ldots+w_{l^*}}{p_{u+1}+\ldots+p_{l^*}}}$$

- If this is not true, we do not worsen the solution by applying the proof of Theorem 4.2.3 and process v after the job sequence u+1,…,l^*
- It is trivial that we can again choose S'' instead of S if it holds S''≤S

Consequences

- Therefore, we know:

$$\frac{w_v}{p_v} < \frac{w_1+\ldots+w_u}{p_1+\ldots+p_u} \wedge \frac{w_v}{p_v} > \frac{w_{u+1}+\ldots+w_{l^*}}{p_{u+1}+\ldots+p_{l^*}}$$

- In addition, l^* is the ρ-factor of (1,…,k). Therefore, it holds:

$$\frac{w_1+\ldots+w_{l^*}}{p_1+\ldots+p_{l^*}} > \frac{w_1+\ldots+w_u}{p_1+\ldots+p_u}$$
$$\Rightarrow \frac{w_{u+1}+\ldots+w_{l^*}}{p_{u+1}+\ldots+p_{l^*}} > \frac{w_1+\ldots+w_u}{p_1+\ldots+p_u}$$

Consequences

$$\Rightarrow \frac{w_{u+1}+\ldots+w_{l^*}}{p_{u+1}+\ldots+p_{l^*}} > \frac{w_1+\ldots+w_u}{p_1+\ldots+p_u} > \frac{w_v}{p_v}$$
$$\text{But we assumed}: \frac{w_v}{p_v} > \frac{w_{u+1}+\ldots+w_{l^*}}{p_{u+1}+\ldots+p_{l^*}}$$

- Therefore, the assumptions of Case 1 and Case 2 together have derived a contradiction
- Therefore, both cases cannot apply simultaneously.
- This obviously completes the proof

Using the result

- The result derived above is intuitive. Its condition implies that the ratios of the total weight divided by the total processing time of the jobs in the string 1,…,l^* must be decreasing in some sense
- If one had decided to start processing a stream, it makes sense to proceed until job l^* is obtained
- By simultaneously using the result derived above, we can use the following algorithm for solving our problem optimally

Solution procedure

4.2.6 Algorithm

Whenever the machine is freed, select among the remaining chains the one with the highest ρ-factor and process this chain without interruption up to the job that determines its ρ-factor. Note that this includes this job itself.

Example

- Consider the following two chains
  - 1 – 2 – 3 – 4
  - 5 – 6 – 7
- The weights and processing times of the jobs are given below

Jobs	1	2	3	4	5	6	7
Weight	6	18	12	8	8	17	18
Processing time	3	6	6	5	4	8	10

Solving the example

- The ρ-factor of the first chain is (6+18)/(3+6)=24/9 and determined by job 2
- The ρ-factor of the second chain is (8+17)/(4+8)=25/12 and determined by job 6
- Therefore, we start processing the first chain (Schedule: 1 – 2)
- The ρ-factor of the remaining first chain is (12)/(6)=2 and determined by job 3
- Therefore, we proceed with the second chain (Schedule: 1 – 2 – 5 – 6)
- The ρ-factor of the remaining second chain is (18)/(10)=1.8 and determined by job 7
- Hence, we proceed with the first chain (Schedule: 1 – 2 – 5 – 6 – 3)
- The ρ-factor of the remaining first chain is (8)/(5) and determined by job 4
- Consequently, we proceed with the second chain (Schedule: 1 – 2 – 5 – 6 – 3 – 7)

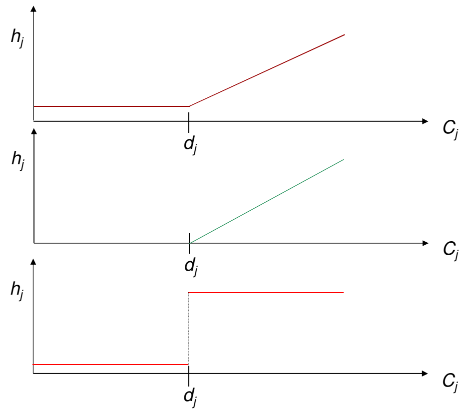
Resulting schedule is 1 – 2 – 5 – 6 – 3 – 7 – 4

Maximum lateness

- In the following, we consider a general penalty function individually defined for a delayed processing of each job
- This leads to a situation where each job j has an individual penalty function $h_j(C_j)$ for a delayed processing of C_j time units
- We assume that each penalty function is monotonous, i.e., the resulting values for increasing completion times are non-decreasing
- In addition, we consider again existing precedence constraints between the different jobs
- Objective function is now:

$$\text{Minimize } \max\{h_1(C_1),\ldots,h_n(C_n)\}$$

Due-date-related penalty functions



Solution procedure

4.2.7 Algorithm

Step 1:

Set $J=\varnothing$; $J^c=\{1,\ldots,n\}$; J' the set of all jobs with no successors

Step 2:

Let j^* be such that $h_{j^*}\left(\sum_{i\in J^c} p_i\right)=\min_{j\in J'}\left(h_j\left(\sum_{i\in J^c} p_i\right)\right)$

Add j^* to J;

Delete j^* from J^c

In order to represent the new set of schedulable jobs, modify J' accordingly

Step 3:

If $J^c=\varnothing$, then stop; otherwise go to Step 2

Consequences

4.2.8 Theorem

Algorithm 4.2.7 attains an optimal schedule for the considered problem

Proof of the Theorem

- Suppose that there exists an iteration where job j^{**} is selected from J' but does not have the minimum completion cost

$$h_{j^*}\left(\sum_{j\in J^c} p_j\right)$$

among the jobs belonging to J' at this moment
- The minimum cost job j^* must then be scheduled at a later iteration, implying that the respective job j^* appears in the sequence before job j^{**}. In addition, some jobs can appear between the jobs j^* and j^{**}

Proof of the Theorem

- In order to complete the proof, we move the sequence position of job j^* just behind job j^{**}.
- What are the consequences for the objective value of the solution?
  - All jobs that are located between j^* and j^{**} in the old schedule S are processed earlier, wherefore the objective function value is not negatively affected
  - What about job j^*? This is the only job whose completion time is increased through the applied modification
  - But we know by assumption that this modified value leads to a smaller penalty function value than the one caused by j^{**} in schedule S
  - Finally, we can state that the value for j^{**} in the new schedule is not increased. Therefore, the maximum of all lateness values in the new schedule is not larger than the objective value of S
- This completes the proof

Special case Z_7

4.2.9 Corollary

For the special case $h_j=\max\{0,C_j-d_j\}$, the application of rule EDD (Earliest Due Date), which schedules the different jobs in a non-decreasing sequence of due dates, results in the optimal solution.

Proof of the Corollary

- In order to prove the claim of the corollary, we can apply Theorem 4.2.8 and Algorithm 4.2.7
- Hence, jobs are scheduled in the first iterations and, therefore, at the end of the arising total sequence with the lowest penalty value
- These values only depend on individual due dates and, therefore, lead to a situation where jobs with the highest due dates are preferred (at the end of the schedule!)
- By preferring the highest due dates for an inverted sequence, we apply the EDD-rule for the original one

Total lateness

- This problem is proven to be *NP*-hard in the ordinary sense, i.e., it exists a pseudo-polynomial time algorithm based on dynamic programming
- However, this problem can be simplified by scheduling jobs which are non-time-critical at the end, i.e., the total processing time of all jobs is lower or equal to their due date

4.3 Sequencing problem with heads and tails

- In what follows, we take a step towards multiple stage problems
- Therefore, we consider a single stage where a scheduling sequence has to be determined. However, each job has preceding and subsequent processes at other stages, which are defined as head and tails
- Consequently, beside p_i, the processing time of the i-th job at the considered stage, there is a head a_i and a tail q_i
- As the pursued objective we consider the minimization of the makespan (lead time)

Deriving a simple lower bound

4.3.1 Lemma

For all subsets I_l of the set of jobs to be processed I, there exists the following lower bound on the optimal cycle time

$$lb(I_l) = \underbrace{\min\{a_i \mid i \in I_l\}}_{a_{\min,I_l}} + \sum_{i \in I_l} p_i + \underbrace{\min\{q_i \mid i \in I_l\}}_{q_{\min,I_l}}$$

Proof of the Lemma

- Consider an arbitrary set of jobs I
- At least $a_{\min,I}$ time units have to elapse before the processing can start
- This processing takes altogether additional $p_{\text{total},I}$ time units
- Finally, there is always one job processed at the last position at the considered stage whose tail increases the total makespan of the processing. And this tail is larger than or equal to $q_{\min,I}$
- Thus, we have shown that the defined sum is a lower bound for the cycle time
- This obviously completes the proof

Complexity of the problem

4.3.2 Lemma

The general scheduling problem with heads and tails is NP-hard

In what follows, we propose the well-known Branch&Bound approach of Carlier (1982) in order to solve the problem optimally

Starting point: The Schrage algorithm

In this greedy approach, we always schedule the ready job with the greatest tail

(i) Set $t = \min_{i \in I} a_i; U = \emptyset; \bar{U} = \{1,...,n\};$
(ii) At time t, schedule amongst the ready jobs i
$(a_i \le t)$ of $\bar{U}$, job j with $q_j = \max\{q_i \mid i \in \bar{U} \wedge a_i \le t\}$
(or any one in the case of ties)
(iii) Set: $U = U \cup \{j\}; \bar{U} = \bar{U} \setminus \{j\}; t_j = t;$
$t = \max\{t_j + p_j, \min_{i \in \bar{U}} a_i\};$
If U is equal to $I = \{1,...,n\}$, the algorithm is finished;
otherwise proceed with step (ii)

Critical path

- The critical path of a solution of the problem always comprises, in the given sequence, the following parts:
  - a head of some job,
  - a sequence of jobs that are iteratively processed – without interruption – at the considered stage, and
  - finally, a tail of some job that is processed at the last position of the critical path
- In what follows, we derive the basic branching rule of the B&B procedure of Schrage by analyzing the critical path of the solution generated by the Schrage procedure

Main result

4.3.3 Theorem

Let L be the makespan of the Schrage schedule

(a). If this schedule is not optimal, there is a critical job c and a critical set J such that:

$$\boxed{lb(J) = \min_{i \in J} a_i + \sum_{i \in J} p_i + \min_{i \in J} q_i > L - p_c}$$

In an optimal schedule, either c is processed before all the jobs of J, or c will be processed after all the jobs of J

(b). If this schedule is optimal, there exists J such that LB(J)=L

Proof of the Theorem

- Let *G* be the disjunctive graph defining the considered problem with source 0 and sink *s*
- In addition, *z* is a critical path passing through a maximal number of jobs
- We modify the numbering of the jobs according to the definition of this path
- Therefore, the jobs processed on this path are numbered from 1 to *p*, i.e., the critical path is (0,1,2,3,…,*p*,*s*)

Proof of the Theorem

- At first, we prove that there is **no processing between the times a_1-1 and a_1**
  - If there is a job processed in this interval, it would be finished at a_1 since the processing of the first job starts just at this point of time
  - If so and there is a job *j* processed there and we ask whether $a_j=t_j$. If so, we can extend the critical path. Obviously, this is not possible due to the assumption of a maximal path *z*
  - However, if $a_j<t_j$ we know due to the processing of the Schrage procedure that there is an additional job processed just before *j*
  - Clearly, because of that cognitions, we know that there is always a final job *k* with $a_k=t_k$. Note that this is at least the job firstly processed in the total schedule
- Hence, we have shown that there is no processing in the interval between a_1-1 and a_1 due to the **maximum choice of *z***

Proof of the Theorem

- Secondly, we show $a_1=\min\{\, a_i \mid 1\le i\le p \,\}$
  - The machine was idle just before job 1 was processed
  - Therefore, the Schrage procedure schedules no job in this interval and job 1 was scheduled subsequently
- I.e., all heads are larger than or equal to the head of job 1
- Thus, we obviously obtain $a_1=\min\{\, a_i \mid 1\le i\le p \,\}$

Proof of the Theorem

- Thirdly, if q_p is the minimal tail of all jobs 1,…,p, the length of the critical path becomes

$$L = a_1 + \sum_{i=1}^{p} p_i + q_p$$
$$= \min\{a_k \mid k\in\{1,\ldots,p\}\} + \sum_{i=1}^{p} p_i + \min\{q_k \mid k\in\{1,\ldots,p\}\}$$

- Hence, the lower bound and the solution value are equal, which immediately proves the optimality of the generated solution

Proof of the Theorem

- But, if otherwise q_p is not the smallest tail of the jobs in {1,…,p}, there is always a job c with largest index whose tail is smaller than q_p
- Let $J=\{c+1,\ldots,p\}$ be the set of subsequently scheduled jobs on the critical path
- We know $q_c<q_r$ for all r in J and additionally $a_r>t_c$ Why?
  - If $a_r\le t_c$, then, owing to its larger tail, job r would be scheduled before job c
  - Hence, we derive $a_r>t_c$

Proof of the Theorem

Consequently, $r\in J$ always implies

$$a_r > t_c = a_1 + p_1 + \ldots + p_{c-1}$$

$\Rightarrow \min\{a_k \mid k\in J\} > a_1 + p_1 + \ldots + p_{c-1}$, and

$q_p = \min\{q_k \mid k\in J\}$ additionally implies

$$\underbrace{\min\{a_k \mid k\in J\} + p_{c+1} + \ldots + p_p + \min\{q_k \mid k\in J\}}_{\text{lower bound of the makespan } lb(J)}$$
$$> \underbrace{a_1 + p_1 + \ldots + p_{c-1} + p_c + p_{c+1} + \ldots + p_p + q_p}_{L} - p_c$$
$$= L - p_c$$

Proof of the Theorem

- Therefore, we have shown that the distance to the lower bound is smaller than p_c
- Thus, what can we learn from this constellation about the searching process?
- Is it necessary to consider constellations where job *c* is processed among set *J*?
  - Answer is NO!
  - Why? If we process *c* somewhere among the jobs of set *J*, the solution considered before cannot be improved since job *c*+1 cannot be processed until the point of time t_c+1 (due to a_{c+1}!). Therefore, the solution is deteriorated by at least one time unit since the position for *p* is optimal according to *J*
- Consequently, we have to decide about the scheduling position of *c* either before or after the set *J*

Branching scheme

- This leads directly to the following branching scheme of the algorithm
- Always proceed with the node resulting in the lowest bound found so far. The Lower Bound of a node *S f*(*S*) is always derived from the maximum of *f*(*F*) (*F*=Father node of *S*), LB(*J*), and LB({*c*} U *J*)
- A new node is added to the tree only if its lower bound is less than the upper bound f_0 found so far
- Apply the Schrage procedure in each node
  - If the solution is optimal, the procedure can be finished and the optimal result is generated
  - Otherwise, compute *c* and *J*
    - Generate the two additional subsequent nodes "*c* before *J*" (=Node 1) and "*c* after *J*" (=Node 2)
    - This can be easily conducted through an aimed modification of the considered instance

Node 1

After determining node *c* and the subsequent set *J*, we modify the tail of *c* in the following way:

$$q_c = \max\left\{q_c, \sum_{r\in J} p_r + q_p\right\}$$

By doing so, the execution of the Schrage procedure always results in a constellation where *c* is processed before all jobs placed in set *J*.

Additionally, the algorithm "knows" the extended tail of c to process this job potentially earlier

Node 2

After determining node *c* and the subsequent set *J*, we modify the head of *c* in the following way:

$$a_c = \max\left\{a_c, \min\{a_r \mid r \in J\} + \sum_{r\in J} p_r\right\}$$

By doing so, the execution of the Schrage procedure always results in a constellation where *c* is processed after all jobs placed in set *J*.

Additionally, the algorithm "knows" the extended head of c to process other jobs potentially earlier

Bound computation and Schrage procedure

- Carlier proposes a specific technique to be able to execute the Schrage procedure in $O(n \cdot \log n)$ time
- **Upper bound computation**: Every time the Schrage procedure is applied, the generated makespan is compared with the current upper bound f_0. Moreover, an alternative constellation conserving the order of all jobs except for job *c*, which is processed after *J*, is additionally compared with this upper bound
- An **additional lower bound** is derived from the application of the Schrage procedure with allowed preemptions

Preemption

- The preemption version of the Schrage algorithm makes use of the greedy rule of the original Schrage procedure, but can additionally preempt each processed job whenever another one arrives with a larger tail
- It is trivial to show that the generated solution is always optimal and is therefore a Lower Bound of the original problem
- In addition, for example, by using heap data structures, this procedure can be executed again in $O(n \cdot \log n)$ steps

Computational results

- This procedure was coded in FORTRAN on an IRIS 80 and initially tested on 1000 problems
- For each problem with *n* jobs, $3 \cdot n$ integers with uniform distributions between 1 and a_{max}, 1 and p_{max} as well as 1 and q_{max} were respectively drawn
- 20 different values for *n* were tested; *n*=50, 100, 150, 200,…, 1.000
- Further details can be found in Carlier (1982)
- 999 problems were solved optimally
- One problem with *n*=850 was not solved (but the distance to bound was 2!). The lower bound was 29.800 (UB=29.802)
- In most cases the solution process takes only a small amount of time (extreme small-sized solution trees)

Branch&Bound of Carlier – Example

- We consider the following example

Jobs i	Job 1	Job 2	Job 3	Job 4	Job 5	Job 6	Job 7
Release dates a_i	10	13	11	20	30	0	30
Processing times p_i	5	6	7	4	3	6	2
Tails q_i	7	26	24	21	8	17	0

Applying Schrage

Nr.	Job	Tail	Start	End	Completed	Av
1	6	17	0	6	23	None
2	**1**	**7**	**10**	**15**	**22**	**2,3**
3	**2**	**26**	**15**	**21**	**47**	**3,4**
4	**3**	**24**	**21**	**28**	**52**	**4**
5	**4**	**21**	**28**	**32**	**53**	**5,7**
6	5	8	32	35	43	7
7	7	0	35	37	37	None

Critical Path: 0-1-2-3-4-*s*

Analyzing the constellation

- c=1 and J={2,3,4}
- LB(J)=min{13,11,20}+6+7+4+min{26,24,21}=11+17+21=49
- LB({1,2,3,4})=10+22+7=39
- UB=53

- Now, we have to branch
  - c before J
  - c after J

Enumeration tree

UB=53
LB=49

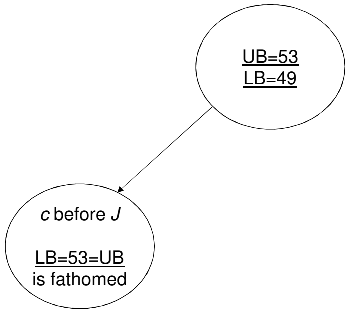
c before *J*

- New problem constellation

Jobs i	Job 1	Job 2	Job 3	Job 4	Job 5	Job 6	Job 7
Release dates a_i	10	13	11	20	30	0	30
Processing times p_i	5	6	7	4	3	6	2
Tails q_i	**17+21=38**	26	24	21	8	17	0

- New lower bound:
  - LB({1,2,3,4})=10+22+21=53
  - Hence, this node can be fathomed

Enumeration tree



c after *J*

- New problem constellation

Jobs i	Job 1	Job 2	Job 3	Job 4	Job 5	Job 6	Job 7
Release dates a_i	11+17 =28	13	11	20	30	0	30
Processing times p_i	5	6	7	4	3	6	2
Tails q_i	7	26	24	21	8	17	0

- New lower bound:
  - LB({1,2,3,4})=11+22+7=40
  - Hence, this node has to be explored

c after *J* – Applying Schrage

Nr.	Job	Tail	Start	End	Completed	Av
1	6	17	0	6	23	None
2	**3**	**24**	**11**	**18**	**42**	**2**
3	**2**	**26**	**18**	**24**	**50**	**4**
4	4	21	24	28	49	1
5	1	7	28	33	40	5,7
6	5	8	33	36	44	7
7	7	0	36	38	38	None

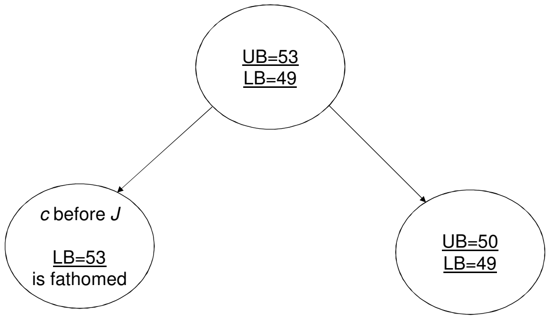
Critical Path: 0-3-2-*s*

Analyzing the constellation

- *c*=3 and *J*={2}
- LB(J)=min{13}+6+min{26}=45
- LB({2,3})=11+6+7+24=48
- Therefore, we inherit the Lower Bound of the father node. This is LB=49
- UB=50

- Now, we have to branch again
  - *c* before *J*
  - *c* after *J*

Enumeration tree



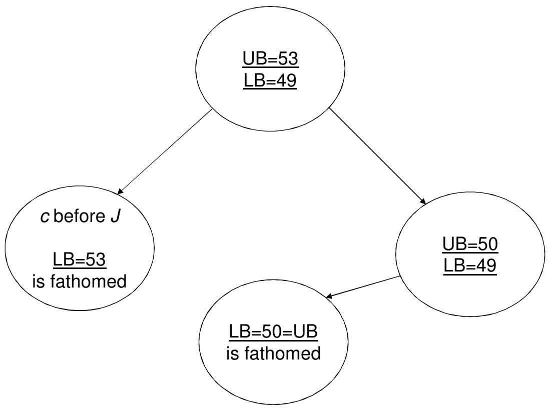
c before J

- New problem constellation

Jobs i	Job 1	Job 2	Job 3	Job 4	Job 5	Job 6	Job 7
Release dates a_i	28	13	11	20	30	0	30
Processing times p_i	5	6	7	4	3	6	2
Tails q_i	7	26	**32**	21	8	17	0

- New lower bound:
  - LB({2,3})=11+13+26=50=UB
  - Hence, this node is fathomed

Enumeration tree



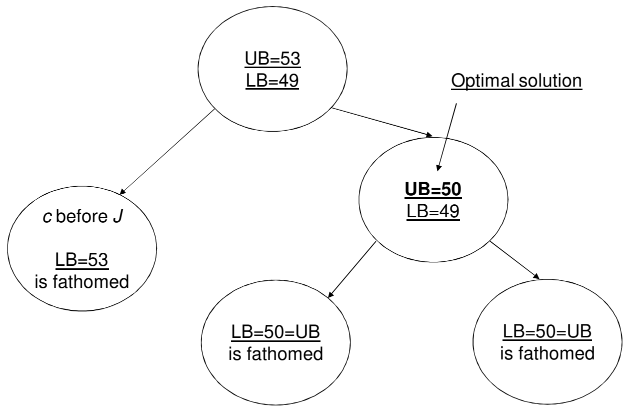
c after J

- New problem constellation

Jobs i	Job 1	Job 2	Job 3	Job 4	Job 5	Job 6	Job 7
Release dates a_i	28	13	**19**	20	30	0	30
Processing times p_i	5	6	7	4	3	6	2
Tails q_i	7	26	24	21	8	17	0

- New lower bound:
  - LB({2,3})=13+13+24=50=UB
  - Hence, this node is fathomed

Enumeration tree



Optimal solution

- Therefore, we obtained an optimal solution
- This optimal solution is given by
  - 6-3-2-4-1-5-7
  - Makespan is: 50

4.4 Multiple stages

- If *M*>1, each objective function itemized in Section 4.2 leads to an *NP*-complete problem for the general job-shop system case
- Therefore, a huge set of different heuristics can be found in literature
- Owing to its simple representation in disjunctive graphs, the minimization of the cycle time or the makespan is frequently pursued
- In comparison to other *NP*-complete problems, the job-shop problem belongs to the most complex ones. This results from the fact that most efficient exact procedures are not able to solve even small-sized problems in a reasonable time (e.g., 10 jobs on 10 machines)

4.4.1 Use of priority rules

- A more intuitive approach can be the application of dynamic rules deciding about the sequence on every machine
- Therefore, in case of an idle machine, this rule decides about the next job to be scheduled by selecting one of the waiting jobs
- Note that this approach is very flexible since it can also be applied to dynamic problems while its complexity only depends on the defined computation of the integrated priority rule
- Frequently, the SPT and its variants integrated into specific hierarchies are applied

4.4.2 Elaborated heuristics

Well-known approaches are for example

- The Shifting Bottleneck Procedure (SBP) of Adams, Balas and Zawack
- The Tabu Search procedure of Nowicki and Smutnicki

4.4.2.1 The Shifting Bottleneck Procedure

- This procedure can be applied to arbitrary *M*-staged job-shop systems to minimize the cycle time
- It makes use of the Branch&Bound algorithm of Carlier as a subroutine
- The problem description is defined as a disjunctive graph
- The bottleneck machine of the total schedule is considered to be planned more accurately in each step

Basic attributes

- The machines are sequenced one at a time, consecutively
- In order to do so, a one-machine scheduling problem with head and tails is optimally solved for each not yet sequenced machine
- This result is taken as rank of the machine to decide about its necessity to sequence it permanently. After sequencing the current machine, all machines sequenced before are resequenced optimally due to the modified heads and tails
- The one-machine problems with head and tails are constructed out of the modified disjunctive graph

Deriving one-machine problems

Let $M_0 \subset M$ be the set of machines that have already been sequenced by choosing selections $S_p\left(p \in M_0\right)$. For any $k \in M \setminus M_0$, let $\left(P\left(k, M_0\right)\right)$ be the problem obtained from the original problem definition replacing each disjunctive arc set $E_p\left(p \in M_0\right)$ by the corresponding selection $S_p\left(p \in M_0\right)$ and deleting each disjunctive arc set $E_p\left(p \in M \setminus M_0, p \neq k\right)$

The procedure

1. M_0=Ø (set of already sequenced machines)
2. Identify a bottleneck machine m among the machines k in $M \backslash M_0$ and sequence it optimally by applying the Carlier algorithm. Set $M_0=M_0 \cup \{m\}$
3. Reoptimize the sequence of each critical machine k in M_0 in turn while keeping the other sequences fixed, i.e., set $M^0=M_0 \backslash \{k\}$ and solve $P(k,M^0)$. Then, if $M_0=M$, stop; otherwise go to step 2

Reoptimization processes

- The reoptimization process is repeated at most three times for sets $|M_0|<|M|$ in every iteration
- Every time a full cycle is completed, the elements of M_0 are reordered according to the non-increasing values of the solutions of the respective one-machine problems with heads and tails
- In the last step, when $|M_0|=|M|$, we continue the local reoptimization process to the point where no more improvement for a full cycle occurs

Two versions

- Two different versions of the SBP are proposed by Balas et al.
  - The first version operates as described above
  - The second one applies the SBP to the nodes of a searching tree generating several solutions simultaneously, i.e., in each branching step, alternative constellations are generated to increase algorithm diversification

The second SBP-version

- The process starts again with the node defined by M_0=Ø
- In each branching step, in a node on level $l=|M_0|$, the $f(l)$ machines with the largest respective machine objective value out of $M \backslash M_0$ are processed as alternative child nodes. Note that f is a monotonous decreasing function reducing the branching degree in the levels that are generated later.
- A second instrument for limiting the size of the branching tree is a penalty function – defined for every node – that penalizes the choices made at different levels in generating the node in question, in proportion to their deviation from the bottleneck, and with a weighting that is heavier for the higher than for the lower levels of the tree. Whenever the value of the penalty function for a node exceeds a predetermined limit, the node is discarded

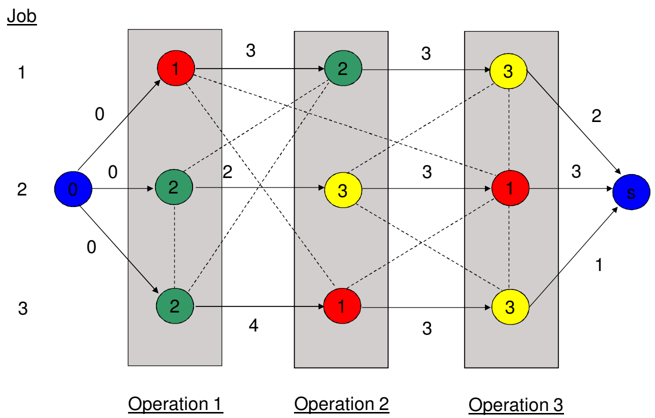
Combined breadth-first / depth-first

- For the first $l^*=M^{1/2}$ levels, the breadth first search is used to produce all successors according to the function f, i.e., a full tree over l^* is generated
- In the second part of the procedure, the nodes are clustered into groups of size $f(l^*)$, containing the successors of level l^*. Subsequently, the depth-first searching phase starts. In this phase, the highest ranking member of one of the groups is chosen and explored straight to the bottom of the search tree, or as far as the penalty function permits
- The current best solution is always stored as an upper bound. Hence, branches, which reach the upper bound, are fathomed. After ending this exploration, the highest rank member of another group of nodes is chosen

Example

$$MS=\begin{pmatrix}1&2&2\\2&3&1\\3&1&3\end{pmatrix};PT=\begin{pmatrix}3&3&3\\3&2&4\\2&3&1\end{pmatrix}$$

Disjunctive graph



One-machine problems

Machine 1			
	Job 1	Job 2	Job 3
Head	0	5	4
Processing time	3	3	3
Tail	5	0	1

One-machine problems

Machine 2			
	Job 1	Job 2	Job 3
Head	3	0	0
Processing time	3	2	4
Tail	2	6	4

One-machine problems

Machine 3			
	Job 1	Job 2	Job 3
Head	6	2	7
Processing time	2	3	1
Tail	0	3	0

Scheduling Machine 1

- Schrage procedure
  - Process job 1 first. Start:0; End:3; Tail:8
  - Process job 3 next. Start:4; End:7; Tail:8
  - Process job 2 at last. Start:7; End:10; Tail:10
  - Objective function value:10

  ➢ Optimal solution since the lower bound is min{5,4}+3+3+min{0,1}=4+6=10

Scheduling Machine 2

- Schrage procedure
  - Process job 2 first. Start:0; End:2; Tail:8
  - Process job 3 next. Start:2; End:6; Tail:10
  - Process job 1 at last. Start:6; End:9; Tail:11
  - Objective function value:11

  - Optimal solution since the lower bound is min{3,0,0}+3+2+4+min{2,6,4}=0+9+2=11

Scheduling Machine 3

- Schrage procedure
  - Process job 2 first. Start:2; End:5; Tail:8
  - Process job 1 next. Start:6; End:8; Tail:8
  - Process job 3 at last. Start:8; End:9; Tail:9
  - Objective function value:9

  - Optimal solution since the lower bound is min{6,7}+2+1+min{0,0}=6+3=9

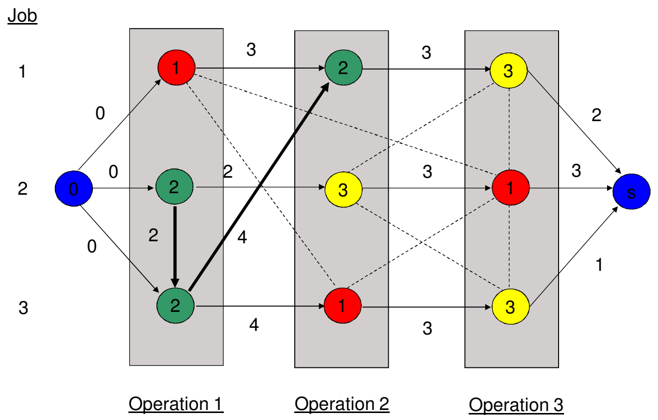
Bottleneck machine

- Machine 1: Completion time is 10
- Machine 2: Completion time is 11
- Machine 3: Completion time is 9

Consequently, the bottleneck machine
… is Machine 2 with Z=11

➢ Therefore, we fix the sequence: 2 – 3 – 1 on this machine

Disjunctive graph



One-machine problems

Machine 1			
	Job 1	Job 2	Job 3
Head	0	5	**6**
Processing time	3	3	3
Tail	5	0	1

One-machine problems

Machine 3			
	Job 1	Job 2	Job 3
Head	**9**	2	**9**
Processing time	2	3	1
Tail	0	3	0

Scheduling Machine 1

- The Schrage procedure provides the following schedule:
  - Process job 1 first. Start:0; End:3; Tail:8
  - Process job 2 next. Start:5; End:8; Tail:8
  - Process job 3 at last. Start:8; End:11; Tail:12
  - Objective function value:12

  - ➢ Cannot be proven to be optimal since the lower bound is min{5,6}+3+3+min{0,1}=5+6=11
  - ➢ *J*={3}, *c*=2;

Modified Branching problem 1
(*c* before *J*)

Machine 1 (**bold** means modified value)			
	Job 1	Job 2 =*c*	Job 3 =*J*
Head	0	5	6
Processing time	3	3	3
Tail	5	**4=3+1**	1

Rescheduling Machine 1
c before *J*

- Schrage procedure
  - Process job 1 first. Start:0; End:3; Tail:8
  - Process job 2 next. Start:5; End:8; Tail:12
  - Process job 3 at last. Start:8; End:11; Tail:11
  - Objective function value:12

  - ➢ Is the optimal solution in the considered sub-tree since the lower bound=min{5,6}+3+3+min{4,1}=5+6+1=12
  - ➢ But already dominated by the solution considered before

Modified Branching problem 1

c after *J*

Machine 1 (**bold** means modified value)	Job 1	Job 2 =*c*	Job 3 =*J*
Head	0	**9=6+3**	6
Processing time	3	3	3
Tail	5	0	1

Rescheduling Machine 1

c after *J*

- Schrage procedure
  - Process job 1 first. Start:0; End:3; Tail:8
  - Process job 3 next. Start:6; End:9; Tail:10
  - Process job 2 at last. Start:9; End:12; Tail:12
  - Objective function value:12

➢ Is the optimal solution in the considered sub-tree since the lower bound amounts to min{9,6}+3+3+min{0,1}=6+6+0=12

Scheduling Machine 3

- Schrage procedure
  - Process job 2 first. Start:2; End:5; Tail:8
  - Process job 1 next. Start:9; End:11; Tail:11
  - Process job 3 at last. Start:11; End:12; Tail:12
  - Objective function value:12

  - Optimal solution since the lower bound is min{9,9}+2+1+min{0,0}=9+3=12

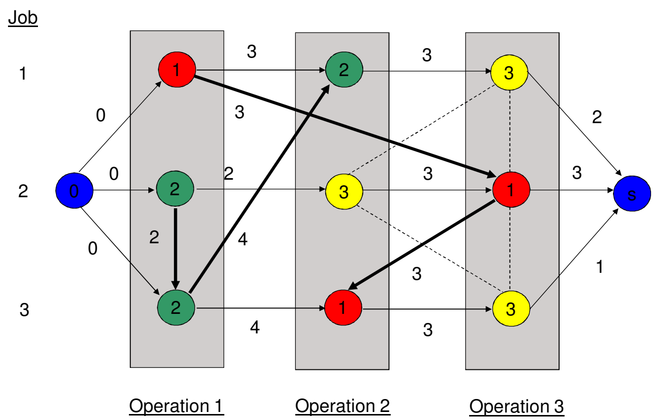
Bottleneck machine

- Machine 1: Completion time is 12
- Machine 3: Completion time is 12

Consequently, the bottleneck machine … is machine 1 with *Z*=12

➢ Therefore, we fix the sequence: 1 – 2 – 3 on this machine

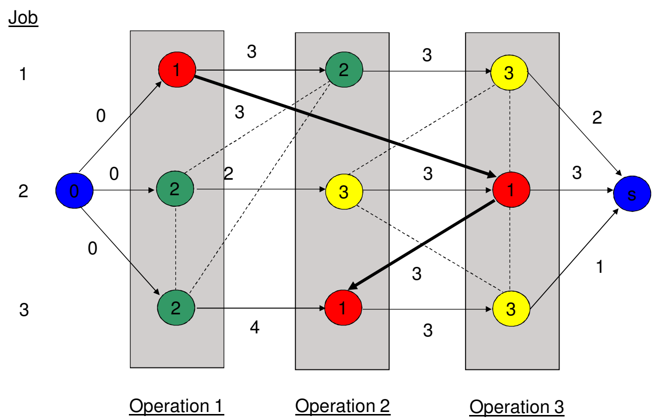
Disjunctive graph



Rescheduling Machine 2

- Now, we have to reoptimize the sequence on Machine 2 according to the potentially modified head and tails
- Therefore, we erase the fixed disjunctive arcs in the graph to derive the modified scheduling problem with head and tails

Disjunctive graph



Modified one-machine problem

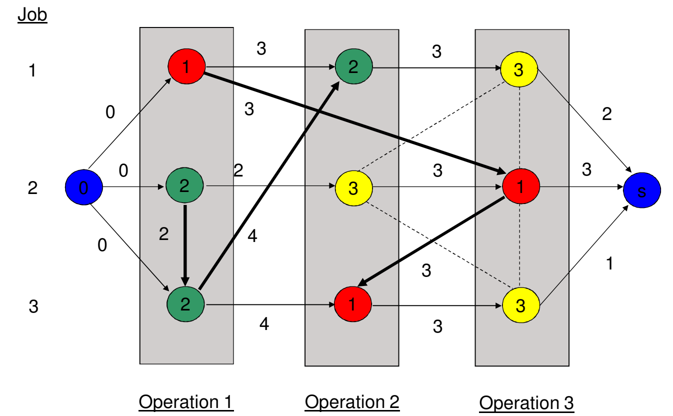
Machine 2 (**bold** means modified value)			
	Job 1	Job 2	Job 3
Head	3	0	0
Processing time	3	2	4
Tail	2	**10**	4

Rescheduling Machine 2

- Schrage procedure
  - Process job 2 first. Start:0; End:2; Tail:12
  - Process job 3 next. Start:2; End:6; Tail:10
  - Process job 1 at last. Start:6; End:9; Tail:11
  - Objective function value:12

  - Optimal solution since the lower bound for set *s*={2} is min{0}+2+min{10}=0+2+10=12
  - The sequence on Machine 2 is kept unchanged!

Disjunctive graph



One-machine problems

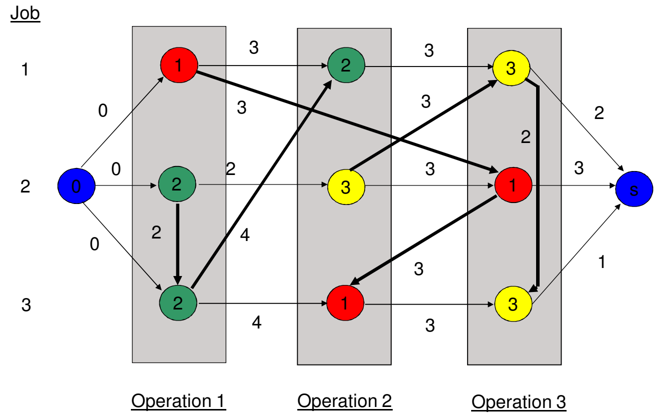
Machine 3 (**bold** means modified value)			
	Job 1	Job 2	Job 3
Head	9	2	**11**
Processing time	2	3	1
Tail	0	**7**	0

Scheduling Machine 3

- Schrage procedure
  - Process job 2 first. Start:2; End:5; Tail:12
  - Process job 1 next. Start:9; End:11; Tail:11
  - Process job 3 at last. Start:11; End:12; Tail:12
  - Objective function value:12

  - Optimal solution since the lower bound is min{9,11}+2+1+min{0,0}=9+3=12
  - Fixing sequence on Machine 3 to 2 – 1 – 3

Disjunctive graph



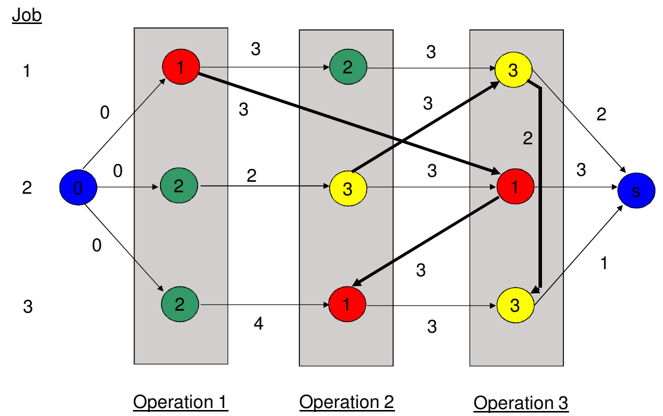
Resequencing

- Subsequently, we have to resequence the already scheduled Machines 1 and 2
- The current objective values of these machines are:
  - Machine 1: 12
  - Machine 2: 12
- We take Machine 2 as the first machine to be rescheduled

Rescheduling Machine 2

- Now, we have to reoptimize the sequence on Machine 2 according to the potentially modified head and tails
- Therefore, we erase the fixed disjunctive arcs in the graph to derive the modified scheduling problem with head and tails

Disjunctive graph



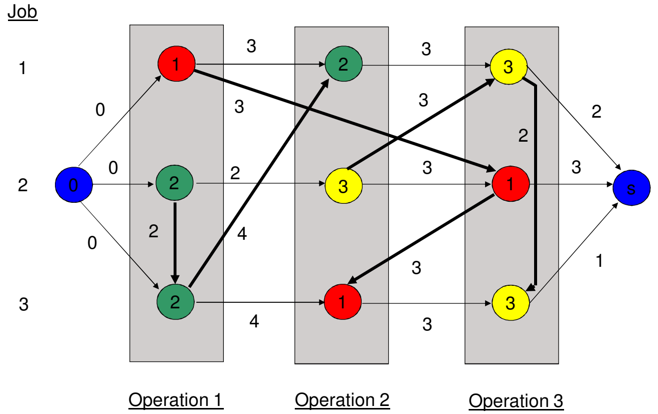
Modified one-machine problem

Machine 2 (**bold** means modified value)			
	Job 1	Job 2	Job 3
Head	3	0	0
Processing time	3	2	4
Tail	**3**	10	4

Rescheduling Machine 2

- Schrage procedure
  - Process job 2 first. Start:0; End:2; Tail:12
  - Process job 3 next. Start:2; End:6; Tail:10
  - Process job 1 at last. Start:6; End:9; Tail:12
  - Objective function value:12
  - Optimal solution since the lower bound for set *s*={2} is min{0}+2+min{10}=0+2+10=12
  - The sequence on Machine 2 is kept unchanged!

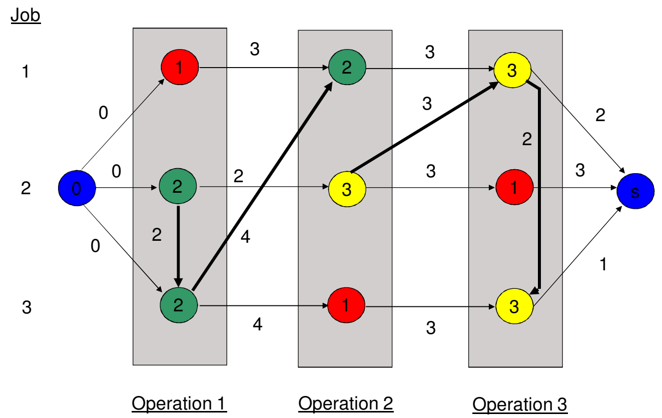
Disjunctive graph



Rescheduling Machine 1

- Now, we have to reoptimize the sequence on Machine 1 according to the potentially modified heads and tails
- Therefore, we erase the fixed disjunctive arcs in the graph to derive the scheduling problem with modified heads and tails

Disjunctive graph



One-machine problems

Machine 1 (**bold** means modified value)			
	Job 1	Job 2	Job 3
Head	0	5	6
Processing time	3	3	3
Tail	**6**	0	1

Rescheduling Machine 1

- Schrage procedure
  - Process job 1 first. Start:0; End:3; Tail:9
  - Process job 2 next. Start:5; End:8; Tail:8
  - Process job 3 at last. Start:8; End:11; Tail:12
  - Objective function value:12

  - Cannot be proven to be optimal since the lower bound is min{5,6}+3+3+min{0,1}=5+6=11
  - J={3}, c=2;

Modified Branching Problem 1

c* before *J

Machine 1 (**bold** means modified value)			
	Job 1	Job 2 =c	Job 3 =J
Head	0	5	6
Processing time	3	3	3
Tail	6	**4=3+1**	1

Rescheduling Machine 1

c **before** *J*

- Schrage procedure
  - Process job 1 first. Start:0; End:3; Tail:9
  - Process job 2 next. Start:5; End:8; Tail:12
  - Process job 3 at last. Start:8; End:11; Tail:11
  - Objective function value:12

  - ➢ Is the optimal solution in the considered sub-tree since the lower bound=min{5,6}+3+3+min{4,1}=5+6+1=12
  - ➢ But already dominated by the solution considered before

Modified Branching Problem 1

c **after** *J*

Machine 1 (**bold** means modified value)			
	Job 1	Job 2 =*c*	Job 3 =*J*
Head	0	**9=6+3**	6
Processing time	3	3	3
Tail	6	0	1

Rescheduling Machine 1

c **after** *J*

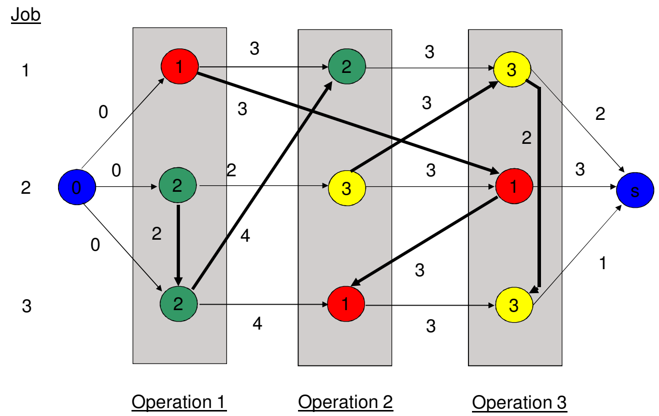
- Schrage procedure
  - Process job 1 first. Start:0; End:3; Tail:9
  - Process job 3 next. Start:6; End:9; Tail:10
  - Process job 2 at last. Start:9; End:12; Tail:12
  - Objective function value:12

  - ➢ Is the optimal solution in the considered sub-tree since the lower bound=min{9,6}+3+3+min{0,1}=6+6+0=12

Rescheduling Machine 1

- The sequence of Machine 1 is kept unchanged!
- 1 – 2 – 3 is the chosen sequence!

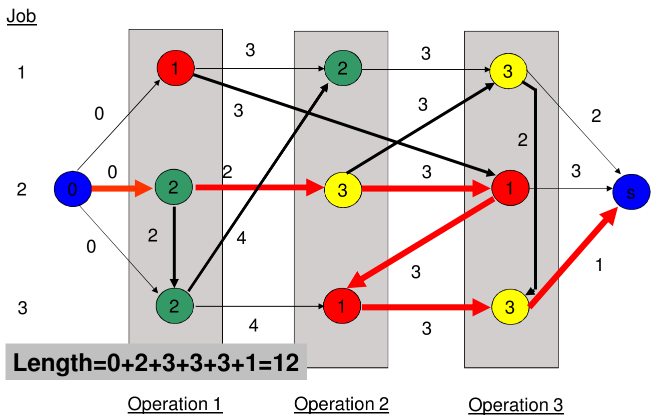
Disjunctive graph of the final solution



Objective function value

- The resulting makespan is determined by the length of the longest path from 0 to *s*
- This path has the total length of 12, which defines the resulting cycle time
- This is illustrated by the final graph

Longest path



Priority rule application

In order to rate the solution quality of the SBP versions, different priority rules are applied in a specific constellation. The applied rules are

- FCFS (=First Come First Serve)
- LST (=Late Start Time)
- EFT (=Early Finish Time)
- LFT (=Late Finish Time)
- MINSLK (=Minimum Slack)
- SPT (=Shortest Processing Time)
- LPT (=Longest Processing Time)
- MIS (=Most Immediate Successors)
- FA (=First Available)
- RANDOM

Computational results

- Procedures were implemented in FORTRAN on a VAX 780/11 on 40 problems taken from well-known benchmarks
- In what follows, we depict the results presented by Balas et al.
- They tested the SBP in its both variants against some simple priority rules
- The consumed CPU time is illustrated in the tables beside the solution quality

Priority rule application

- First, the priority rule algorithms are applied in a straightforward fashion
- Second, the priority rules are applied in a random fashion by applying all rules
  - The randomized rule is to select one of the available operations to be processed next randomly
  - This is done by applying a probability distribution which makes the odds of being selected proportional to the priority assigned to each operation by the given dispatching rule
  - The run is repeated until ten consecutive runs produce no improvement, and the best result obtained is reported as the procedure's output

Performance results

	Number of			SBI			SBII			
Instance	Machines	Jobs	Operations	Value	CPU Sec	Micro-runs	Value	CPU Sec	Macro-runs	LB
1	5	4	20	13*	0.50	21	---	---	---	13
2	6	6	36	55*	1.50	82	---	---	---	52
3	10	10	100	1015	10.10	249	930*#	851	270	808
4	5	20	100	1290	3.50	71	1178	80	32	1164
5	10	10	100	1306	5.70	181	1239	1503	352	1028
6	10	10	100	962	12.67	235	943	1101	343	835
7	15	20	300	730	118.87	1057	710	1269	30	650
8	15	20	300	774	125.02	1105	716	1775	35	597
9	15	20	300	751	94.32	845	735	1312	35	616
10	10	15	150	1172	21.89	343	1084	362	25	995
11	10	15	150	1040	19.24	293	994	414	44	913
12	10	20	200	1304	48.54	525	1224	744	62	1218
13	10	20	200	1325	45.54	434	1291	837	64	1235
14	10	30	300	1784*	38.26	212	---	---	---	1784
15	10	30	300	1850*	29.06	164	---	---	---	1850
16	10	40	400	2553*	11.05	61	---	---	---	2553
17	10	40	400	2228*	75.03	226	---	---	---	2228
18	10	50	500	2864*	53.42	98	---	---	---	2864
19	10	50	500	2985*	27.47	75	---	---	---	2985

Value: makespan of the best schedule obtained
Micro-runs: number of the one-machine problems solved
Macro-runs: number of times SBI was run

*: value known to be optimal
#: optimal value found after 320 seconds
LB: lower bound given by solution value for the first bottleneck problem

Performance results with 5 machines

	Priority Dispatching Rule				SBI		SBII			Improvement	
	Straight		Randomized		Value	CPU Sec	Value	CPU Sec	LB	SBI %	SBII %
Problem	Value	CPU Sec	Value	CPU Sec							
5 machines, 10 jobs											
1	679	4.11	679	157	666*	1.26	---	---	666	1.91	---
2	792	4.03	727	125	720	1.69	669	12.5	655	1.0	7.98
3	673	4.22	634	113	623	2.46	605	31.8	588	1.74	4.57
4	670	4.33	621	139	597	2.79	593	45.4	567	3.86	4.56
5	594	3.58	594	100	593*	0.52	---	---	593	0.2	---
5 machines, 15 jobs											
6	927	8.20	927	233	926*	1.28	---	---	926	0	---
7	947	8.57	920	194	890*	1.51	---	---	890	3.26	---
8	880	8.28	866	280	868	2.41	863*	4.52	863	-0.2	0.35
9	952	8.31	952	260	951*	0.85	---	---	951	0.14	---
10	959*	8.40	959*	217	959*	0.81	---	---	958	0	---
5 machines, 20 jobs											
11	1223	15.24	1223	364	1222*	2.03	---	---	---	0	---
12	1041	12.68	1040	291	1039*	0.87	---	---	---	0	---
13	1151	14.17	1151	409	1150*	1.23	---	---	---	0	---
14	1293	14.77	1293	379	1292*	0.94	---	---	---	0	---
15	1320	15.74	1314	327	1207*	3.09	---	---	---	8.14	---

Performance results with 10 machines

	Priority Dispatching Rule				SBI		SBII			Improvement	
	Straight		Randomized		Value	CPU Sec	Value	CPU Sec	LB	SBI %	SBII %
Problem	Value	CPU Sec	Value	CPU Sec							
10 machines, 10 jobs											
16	1036	7.66	1036	240	1021	6.48	978	240**	875	1.45	5.60
17	857	6.85	857	192	796	4.58	787	192**	737	7.12	8.17
18	673	6.55	897	225	891	10.2	859	225**	770	0.67	4.24
19	670	7.45	898	240	875	7.40	860	240**	709	2.56	4.24
20	594	7.89	942	289	924	10.2	914	289**	807	1.91	2.97
10 machines, 15 jobs											
21	1208	14.71	1198	362	1172	21.9	1084	362**	995	2.17	9.52
22	1085	13.93	1038	414	1040	19.2	944	419**	913	-0.2	9.06
23	1163	14.22	1108	417	1061	24.6	1032*	225**	1023	4.24	6.86
24	1142	14.33	1048	435	1000	25.5	976	434**	881	4.58	6.87
25	1259	14.70	1160	430	1048	27.9	1017	430**	894	1.03	3.71
10 machines, 20 jobs											
26	1373	24.62	1373	744	1304	48.5	1224	744**	1218	5.03	10.85
27	1472	25.79	1417	837	1325	45.5	1291	837**	1235	6.49	8.89
28	1475	25.5	1402	901	1256	28.5	1250	901**	1216	10.41	10.84
29	1539	25.38	1382	892	1294	48.0	1239	892**	1114	6.37	10.35
30	1604	26.7	1508	816	1403	37.8	1355*	551**	1355	6.96	10.15

Further performance results

	Priority Dispatching Rule				SBI		SBII			Improvement	
	Straight		Randomized		Value	CPU Sec	Value	CPU Sec	LB	SBI %	SBII %
Problem	Value	CPU Sec	Value	CPU Sec							
10 machines, 30 jobs											
31	1935	55.42	1852	1786	1784*	38.3	---	---	---	3.67	---
32	1969	57.48	1916	1889	1850*	29.1	---	---	---	3.44	---
33	1871	54.13	1806	1313	1719*	25.6	---	---	---	4.82	---
34	1926	55.65	1844	1559	1721*	27.6	---	---	---	6.67	---
35	1997	56.61	1987	1537	1888*	21.3	---	---	---	4.98	---
15 machines, 15 jobs											
36	1517	26.20	1385	735	1351	46.9	1305	735**	1224	2.45	5.78
37	1670	26.95	1551	837	1485	61.4	1423	837**	1355	4.26	8.25
38	1405	24.43	1388	1079	1280	57.7	1255	1079**	1077	7.78	9.58
39	1436	24.40	1341	669	1321	71.8	1273	669**	1221	1.49	5.07
40	1477	24,71	1383	899	1326	76.7	1269	899**	1170	4.12	8.24

Value: makespan of the best schedule obtained
LB: lower bound given by solution value for the first bottleneck problem

Improvement: improvement (in percent) in solution value over that found by the randomized priority dispatching rule
*: value proved to be optimal
**: time limit set to time required by randomized priority dispatching rule.

Main results

- Priority rules:
  - No domination between the rules can be identified
  - Eight of the ten rules showed best result on at least one problem
  - Two rules (LPT and FA) never

- Priority rules vs. SBP I/II
  - In 38 cases SBP I finds better solutions than the constellations generated by the priority rule procedure whether in the straight or randomized version
  - Furthermore, Version 2 finds substantially improved solutions for many constellations most of the time
  - Altogether, it can be stated that SBP II is always – without exception – at least as good as the randomized priority rule
  - Moreover, in the vast majority of the considered cases, it is considerably better
  - Typical average improvement rates were between 4 and 10 percent

SBP – Pros and Cons

- **Pros**
  - Elaborated procedure
  - Despite the fact that the procedure uses a Branch&Bound procedure to tackle an *NP*-hard problem as a frequently called subroutine, it is quite fast in comparison to well-known meta strategies, as for example, the Tabu Search procedure of Nowicki and Smutnicki
  - SBP I is frequently used as an initial procedure to generate a first solution with quite good quality

- **Cons**
  - Solution quality is poorer than known from elaborated meta strategies
  - Single priority rules are much faster

4.4.2.2 Tabu Search by Nowicki-Smutnicki

- Besides the SBP as well as various Branch&Bound-procedures, meta heuristics have recently been developed for the job-shop scheduling problem with makespan objective
- A very efficient and relatively easy to implement algorithm is the **Tabu Search** *(TS)* procedure introduced by Nowicki and Smutnicki in 1996
- The algorithm bases on the disjunctive graph and tries to reduce the problems makespan iteratively by changing the job sequence within the critical path

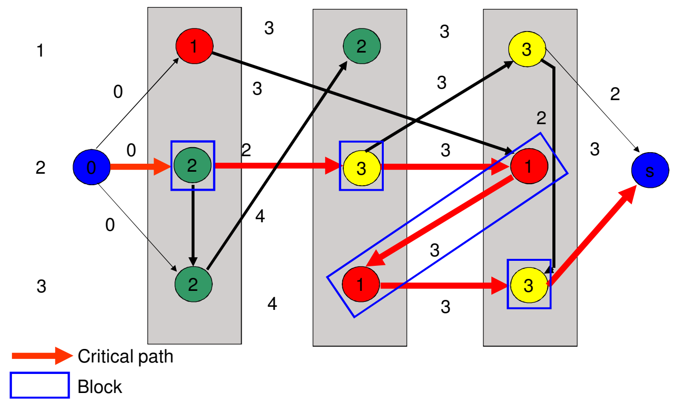
Neighborhood Search

- Tabu Search methods are based on **Neighborhood Search**, a local search method.
- Given a solution s, a Neighborhood Search creates out of a solution π a new solution π' by manipulating π; this operation is called a **move**.
- The set of moves applicable on a given solution s is called the neighborhood $N(\pi)$. Neighborhood Search selects the best move in $N(\pi)$ and applies it.
- If a solution can be represented as a permutation of numbers, common Neighborhood Search moves are swaps and shifts within this permutation.

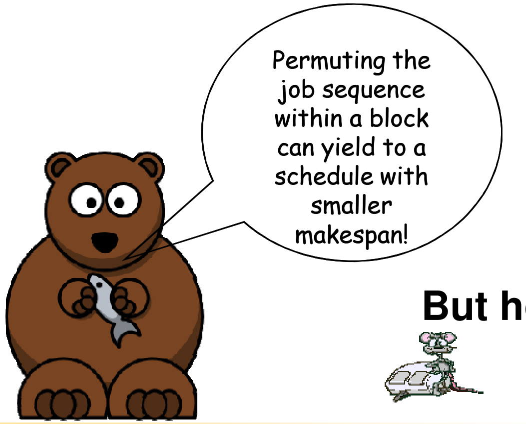
Critical path

- Let $u = (u_1,\ldots,u_w)$ denote the critical path, with w the number of operations on a longest path within the directed disjunctive graph
- The path can be divided into **blocks** $B_1, \ldots, B_r$ with the following attributes:
  - $B_i = (u_{a_i}, u_{a_i+1}, \ldots, u_{b_i})$ and $1=a_1 \le b_1 < b_1+1=a_2 \le b_2 < b_2+1=a_3 \le \ldots \le a_r \le b_r = w$
  - B_i contains all operations processed on the same machine $(i = 1, \ldots, r)$
  - Two consecutive blocks contain operations processed on different machines, *i.e.*, $\mu(B_i) \neq \mu(B_{i+1}), i = 1, \ldots, r-1$

Critical path and block representation



Idea



But how?

The applied neighborhood

- The size of the neighborhood plays an important role.
- Thus, Nowicki and Smutnicki introduced a reduced neighborhood with the following moves:

- In block B_1 the last two operations are permutated
- In block B_2 to B_{r-1} the first two operations and, if $a_i<b_i$, the last two operations are permuted
- In block B_r the first two operations are permutated

Mathematical representation of the moves

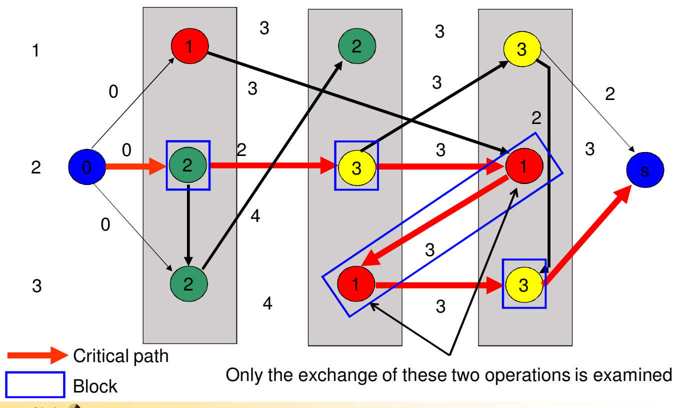
Let $V(\pi) = (V_1(\pi),\ldots, V_r(\pi))$ denote the set of moves that are applicable to a given job sequence π.

$$V_1(\pi)=\begin{cases}\{(u_{b_1-1},u_{b_1})\} & \text{if } a_1<b_1 \text{ and } r>1\\ \varnothing & \text{else}\end{cases}$$

$$V_i(\pi)=\begin{cases}\{(u_{a_i},u_{a_i+1}),(u_{b_i-1},u_{b_i})\} & \text{if } a_i<b_i\\ \varnothing & \text{else}\end{cases}$$

$$V_r(\pi)=\begin{cases}\{(u_{a_r},u_{a_r+1})\} & \text{if } a_r<b_r \text{ and } r>1\\ \varnothing & \text{else}\end{cases}$$

Applied to our example



Remark to the critical path

One might argue that the critical path is not well-defined. But numerical results showed that the selection of one critical path has a minor influence in regard to the solutions quality.

➔ An arbitrary critical path can be chosen.

Tabu List

- A major drawback of local search procedures, such as hill-climbing, is cycling between two solutions and only returning a local optimal solution
- In order to avoid cycling within the searching process, Tabu Search algorithms use a short time memory of blocked moves, called **Tabu List**
- If a move $v=(x,y)$ is performed, the inverse move $v'=(y,x)$ is added to the Tabu List
- The Tabu List has a given size *maxt*, and it contains the inverse moves of the moves applied in the last *maxt* iterations

Aspiration criterion

- To secure that promising moves are not blocked, Nowicki and Smutnicki divide the set operations in the Tabu List into subsets *UP* and *UNP*
- *UP* contains all blocked moves leading to a better solution than the ones visited in all past iterations (profitable moves)
- *UNP* consists of all blocked non-profitable moves
- A criterion, called aspiration criterion, allows the search to perform a profitable move although it belongs to the Tabu List.

Long time memory

- If a solution possesses a good objective function value, it is likely that its neighbors contain good objective function values as well
- Since from a given solution with small objective function value only the best move was chosen, the observation of other neighbors was discarded although they could guide into regions with good solutions, too
- To take this thought into account, Nowicki and Smutnicki proposed to embed their procedure into a guided super routine by storing the solutions with the lowest objective function values within a list L
- The elements of L consist of the permutation π for the given solution, a modified neighborhood $N(\pi)\backslash\{v'\}$ (v' is the already applied move), and the Tabu List T

Performance analysis – test sets

- Nowicki and Smutnicki tested their algorithms on groups of well-known job-shop scheduling instances
  - Group I: 45 instances with 36 to 100 operations
  - Group II:
    - 80 instances with 225 to 2000 operations
    - 40 instances created by a random generator with 2500 to 10000 operations

Performance analysis - results

Test set		Number of instances	C^* better than best-known value	Optimality proven
Group I		45	30	In 20 of 30 unknown cases*
Group II	a)	80	33	In 10 of 61 unknown cases*
	b)	40	No references available	No references available

*Unknown up to Nowicki and Smutnicki (1996)

Nowicki and Smutnicki – Summary

- The procedure is a solution method for solving the job-shop scheduling problem with makespan objective which is relatively easy
- It substantially intensifies the searching process in promising regions, evaluates the neighborhoods in the single steps very fast, and the authors can improve the best known makespan for difficult problem instances in many cases
- In 2005, Nowicki and Smutnicki propose a further advanced Tabu Search procedure with improved diversification (long term behavior)
- Additionally, they propose an improved starting heuristic in order to construct a suitable starting solution

4.5 Flow-shop problems

- In the following, we consider flow-shop problems as a special case of job-shop systems
- In this special case, each job has an identical machine sequence in which it is processed
- Therefore, we can define a definite numbering $(1,\ldots,M)$ of the used resources that determine the processing sequence of each job
- Despite the fact that the total solution space still consists of altogether $(N!)^M$ constellations, this problem seems to be somehow relaxed in comparison to the general job-shop problem

The dominance criteria

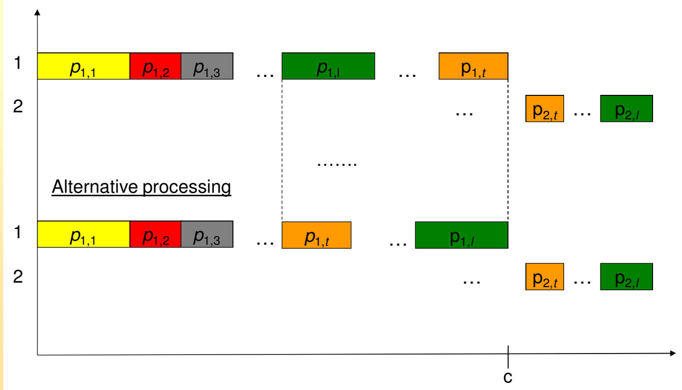
The first dominance criterion:

In an M-staged flow-shop system, there is always an optimal solution minimizing the makespan where the scheduling on the first two machines is identical. This is also true for the minimization of the total lead time.

Proof of the first dominance criterion

- Let us assume there are unequal sequences that are processed on the first two machines
- Let 1 – 2 – 3 – … – *N* be the job sequence applied to machine 2
- We define t as the first (lowest numbered) position in this job sequence where a difference between the sequences on machine 1 and machine 2 arises
- Therefore, we have the sequence 1 – 2 – 3 – … – *t*-1 – *l* with *l*>*t* at the first stage
- In what follows, we consider an alternative constellation by exchanging *t* and *l* on machine 1

Illustration



Consequences for jobs

- **Job t**
  - The exchange on machine 1 can improve only the subsequent constellation by an earlier processing at stage 2
  - Therefore, the remaining schedule is of better or at least of an equal quality
- **Job l**
  - Firstly, note that the end of processing of job *l* at stage 1 in the modified constellation is equal to the point of time the processing of job *t* ends at stage 1 in the original constellation. Let *c* denote this point of time in both schedules
  - Therefore, job *l* can not be processed at stage 2 before *t* is processed. Note that this applies to both schedules
    - Schedule 1: Reason: Processing of job *t* at stage 2 before job *l*. In addition, job *t* at stage 2 has to wait for its processing at stage 1, which is not ended before *c*
    - Schedule 2: Reason: Processing of job *l* at stage 1. In detail, job *l* at stage 2 has to wait for its processing at stage 1, which is not ended before *c*

Conclusions

- The processing times of job *t* and job *l* at stage 2 are not influenced by the executed exchange
- No effect on the total makespan as well as on the resulting cycle time
- This completes the proof

The dominance criteria

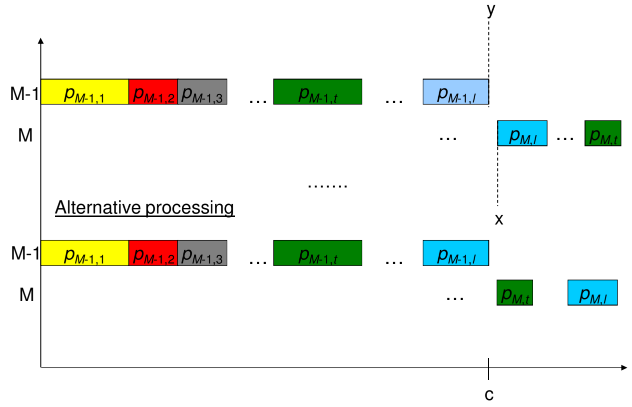
The second dominance criterion:

In an M-staged flow-shop system there is always an optimal solution generating the minimal makespan where the scheduling on the last two machines is identical

Proof of the second dominance criterion

- Again, we assume that there is no equal sequence at the last two stages
- In detail, we assume that the sequence 1 – 2 – 3 – … – *N* is processed on machine *M*-1
- Let *t* be the minimal number of a job where the sequences at stages *M*-1 and *M* differ, i.e., 1 – 2 – 3 – … – *l*, with *l*>*t* is processed on machine *M*
- Now, we consider an alternative constellation where we exchange job *l* and job *t* at the last stage

Illustration



Consequences

- Optimal Schedule
  - Let x be the beginning of the processing of job l at stage M while y denotes the end of processing of job l at stage M-1
  - We know x≥y
  - We have cycle time C
- Alternative Schedule
  - In order to compute the makespan for this schedule, we know that nothing is lost due to the executed exchange of l and t
  - Moreover, there is no side-effect on the jobs processed after job t, wherefore the new cycle time C' is lower than or equal to c
- This completes the proof

4.5.1 The procedure of Johnson

- In what follows, we consider the special case of the flow-shop problem where only production stages are given
- Note that already this quite simple constellation is an NP-complete problem for the general job-shop case
- In contrast to this, we will present a very efficient algorithm generating an optimal schedule for the 2-staged flow-shop problem in O(N log N) steps: the so-called **Johnson algorithm**. This algorithm determines the schedule with the minimal makespan

Changing the sequence in 2/3-FS

4.5.1.1 Theorem

For the two- or three-staged flow-shop with the objective of makespan minimization, there is always an optimal solution that has equal scheduling sequences on all machines

Proof of the Theorem

- Clearly, we may apply the two dominance criteria
- Thus, the claim of the Theorem follows immediately as a corollary of both criteria
- This completes the proof

4.5.1.2 Algorithm (Johnson)

Initialization: $R = \{1,2,3,\ldots,N\}$ is the list of all jobs to be scheduled

1. Determine job $\hat{n}$ by:
   $\min\{p_{1,\hat{n}}, p_{2,\hat{n}}\} = \min\{p_{1,n}, p_{2,n} \mid 1 \le n \le N\}$
2. If $p_{1,\hat{n}} = \min\{p_{1,\hat{n}}, p_{2,\hat{n}}\}$,
   then $\hat{n}$ is placed on the next available position at the top of the current schedule;
   otherwise $\hat{n}$ is placed on the next available position at the end of the current schedule
3. Delete $\hat{n}$ from the list of jobs to be processed in the schedule
4. Proceed with step 1 until the list of remaining jobs R is empty

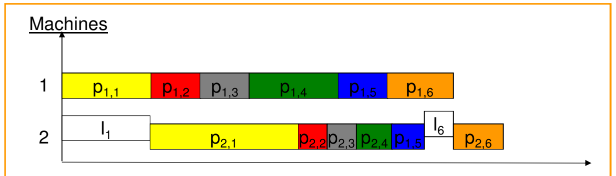
The proof of optimality

4.5.1.3 Theorem

Algorithm 4.5.1.2 generates an optimal solution for the makespan minimization problem in a two-staged flow-shop production system

Proof of the Theorem

- First of all, we have to introduce some additional parameters to determine how the makespan is affected by the chosen schedule
  - Since processing of the first machine is never a bottleneck, we concentrate on the second one
  - Idle times and the total processing time at this stage determine the sought makespan
  - Therefore, I_j should determine the amount of idle times on machine 2 before processing job j, i.e., if the machine was not idle, the parameter is set to zero
  - Note that these values depend on the generated solution while the total processing time at stage 2 is always fixed



Calculation of the occurring idle times

$$I_1 = p_{1,1}$$
$$I_2 = \max\{p_{1,1} + p_{1,2} - I_1 - p_{2,1}, 0\}$$
$$I_3 = \max\{p_{1,1} + p_{1,2} + p_{1,3} - I_1 - I_2 - p_{2,1} - p_{2,2}, 0\}$$
$$\ldots$$
$$\Rightarrow \forall j \in \{1,\ldots,N\}: I_j = \max\left\{\sum_{i=1}^{j} p_{1,i} - \sum_{i=1}^{j-1} I_i - \sum_{i=1}^{j-1} p_{2,i}, 0\right\}$$

The objective value

$$Z = \underbrace{\sum_{i=1}^{N} p_{2,i}}_{\text{Constant production time at stage 2}} + \underbrace{\sum_{i=1}^{N} I_i}_{\text{Additional solution-dependent waiting time at stage 2}}$$

First, we have to compute the total sum of waiting times, i.e.,

$$\sum_{i=1}^{N} I_i$$

Computing the total sum of waiting times

$$\sum_{i=1}^{N} I_i = \sum_{i=1}^{N} \max\left\{\sum_{k=1}^{i} p_{1,k} - \sum_{k=1}^{i-1} I_k - \sum_{k=1}^{i-1} p_{2,k}, 0\right\}$$

Lemma :

$$\sum_{i=1}^{N} I_i = \max\left\{\sum_{i=1}^{k} p_{1,i} - \sum_{i=1}^{k-1} p_{2,i} \mid 1 \le k \le N\right\}$$

Proof of the Lemma

We show the claim by induction:

Start: $N = 1$

$$\sum_{i=1}^{1} I_i = \sum_{i=1}^{1} \max\left\{\sum_{k=1}^{i} p_{1,k} - \sum_{k=1}^{i-1} I_k - \sum_{k=1}^{i-1} p_{2,k}, 0\right\} = \max\left\{\sum_{k=1}^{1} p_{1,k} - \sum_{k=1}^{0} I_k - \sum_{k=1}^{0} p_{2,k}, 0\right\}$$

$$= \max\{p_{1,1}, 0\} = p_{1,1}$$

$N \to N+1$

It holds: $\sum_{i=1}^{N} I_i = \max\left\{\sum_{i=1}^{k} p_{1,i} - \sum_{i=1}^{k-1} p_{2,i} \mid 1 \le k \le N\right\}$

Proof of the Lemma

We compute

$$\sum_{i=1}^{N+1} I_i = \sum_{i=1}^{N} I_i + \max\left\{\sum_{i=1}^{N+1} p_{1,i} - \sum_{i=1}^{N} I_i - \sum_{i=1}^{N} p_{2,i}, 0\right\}$$

$$= \max\left\{\sum_{i=1}^{k} p_{1,i} - \sum_{i=1}^{k-1} p_{2,i} \mid 1 \le k \le N\right\} + \max\left\{\sum_{i=1}^{N+1} p_{1,i} - \sum_{i=1}^{N} I_i - \sum_{i=1}^{N} p_{2,i}, 0\right\}$$

Proof of the Lemma

Case 1:

$$\sum_{i=1}^{N+1} p_{1,i} - \sum_{i=1}^{N} p_{2,i} > \sum_{i=1}^{N} I_i$$

$\Rightarrow$ The difference $\sum_{i=1}^{N+1} p_{1,i} - \sum_{i=1}^{N} p_{2,i} - \sum_{i=1}^{N} I_i$ is added while the resulting value is equal to $\max\left\{\sum_{i=1}^{k} p_{1,i} - \sum_{i=1}^{k-1} p_{2,i} \mid 1 \le k \le N+1\right\}$ since $\sum_{i=1}^{N+1} p_{1,i} - \sum_{i=1}^{N} p_{2,i}$ is the new maximum. Therefore, the lemma gives the correct calculation in this case.

Proof of the Lemma

Case 2:

$$\sum_{i=1}^{N+1} p_{1,i} - \sum_{i=1}^{N} p_{2,i} \le \sum_{i=1}^{N} I_i$$

$\Rightarrow$ The difference $\sum_{i=1}^{N+1} p_{1,i} - \sum_{i=1}^{N} p_{2,i} - \sum_{i=1}^{N} I_i$ is at most zero and, therefore, zero is added to the sum. In addition, it holds:

$\sum_{i=1}^{N+1} p_{1,i} - \sum_{i=1}^{N} p_{2,i} \le \max\left\{\sum_{i=1}^{k} p_{1,i} - \sum_{i=1}^{k-1} p_{2,i} \mid 1 \le k \le N\right\}$, wherefore the total sum is equal to $\max\left\{\sum_{i=1}^{k} p_{1,i} - \sum_{i=1}^{k-1} p_{2,i} \mid 1 \le k \le N+1\right\}$.

Therefore, the lemma gives the correct calculation also in this second case .

Intermediate summary

- A found solution can only influence waiting times on machine 2 to minimize the makespan
- We have generated a compact form for computing all resulting waiting times on machine 2

Notation

In what follows, we make use of the following additional parameters:

$$\forall k \in \{1,...,N\}: Y_k = \sum_{i=1}^{k} p_{1,i} - \sum_{i=1}^{k-1} p_{2,i}$$

$$\Rightarrow$$

$$\forall k \in \{1,...,N\}: \sum_{i=1}^{k} I_i = \max\{Y_t \mid 1 \le t \le N\}$$

Transformation

- Now, the preliminary work is done to start the proof of the theorem by transforming an optimal solution into a new one respecting the claimed attributes
- Therefore, let us assume we have an optimal schedule S found with sequence (1 – 2 – 3 – … – N)
- Furthermore, let us assume we have a solution not fulfilling the construction rules of the algorithm of Johnson
- If so, there is a minimally chosen index i with:

$$\min\{p_{1,i}, p_{2,i+1}\} > \min\{p_{2,i}, p_{1,i+1}\}$$

Transformation

- Note that in the algorithm of Johnson it would have been processed after i+1
- Now, we generate schedule T out of S by exchanging the jobs i and i+1
- Owing to this simple modification, we can easily derive the new updated objective function by computing the new Y_k-values

Comparing schedules S and T

Let us consider Y_k^T as the value for job *k* under schedule *T*

$$Y_i = \sum_{k=1}^{i} p_{1,k} - \sum_{k=1}^{i-1} p_{2,k} \qquad (1 \to 2 \to \ldots \to i-1 \to i \to i+1 \to \ldots \to N)$$

$$Y_i^T = \sum_{k=1}^{i+1} p_{1,k} - \sum_{k=1}^{i-1} p_{2,k} - p_{2,i+1} \qquad (1 \to 2 \to \ldots \to i-1 \to i+1 \to i \to \ldots \to N)$$

$$Y_{i+1} = \sum_{k=1}^{i+1} p_{1,k} - \sum_{k=1}^{i} p_{2,k}$$

$$Y_{i+1}^T = \sum_{k=1}^{i-1} p_{1,k} + p_{1,i+1} - \sum_{k=1}^{i-1} p_{2,k}$$

All other values are not affected !

Comparing schedules S and T

$$\Rightarrow \max\{Y_i^T, Y_{i+1}^T\} = \max\left\{\sum_{k=1}^{i+1} p_{1,k} - \sum_{k=1}^{i-1} p_{2,k} - p_{2,i+1}, \sum_{k=1}^{i-1} p_{1,k} + p_{1,i+1} - \sum_{k=1}^{i-1} p_{2,k}\right\}$$

$$= \sum_{k=1}^{i-1} p_{1,k} - \sum_{k=1}^{i-1} p_{2,k} + \max\{p_{1,i} + p_{1,i+1} - p_{2,i+1}, p_{1,i+1}\}$$

$$= \sum_{k=1}^{i-1} p_{1,k} - \sum_{k=1}^{i-1} p_{2,k} + p_{1,i+1} + \max\{p_{1,i} - p_{2,i+1}, 0\}$$

$$\Rightarrow \max\{Y_i, Y_{i+1}\} = \max\left\{\sum_{k=1}^{i} p_{1,k} - \sum_{k=1}^{i-1} p_{2,k}, \sum_{k=1}^{i+1} p_{1,k} - \sum_{k=1}^{i} p_{2,k}\right\}$$

$$= \sum_{k=1}^{i-1} p_{1,k} - \sum_{k=1}^{i-1} p_{2,k} + \max\{p_{1,i}, p_{1,i} + p_{1,i+1} - p_{2,i}\}$$

$$= \sum_{k=1}^{i-1} p_{1,k} - \sum_{k=1}^{i-1} p_{2,k} + p_{1,i} + \max\{0, p_{1,i+1} - p_{2,i}\}$$

Comparing schedules S and T – Case 1

$$T: \sum_{k=1}^{i-1} p_{1,k} - \sum_{k=1}^{i-1} p_{2,k} + p_{1,i+1} + \max\{p_{1,i} - p_{2,i+1}, 0\}$$

$$S: \sum_{k=1}^{i-1} p_{1,k} - \sum_{k=1}^{i-1} p_{2,k} + p_{1,i} + \max\{0, p_{1,i+1} - p_{2,i}\}$$

We know: $\min\{p_{1,i}, p_{2,i+1}\} > \min\{p_{2,i}, p_{1,i+1}\}$

Case 1: $p_{2,i+1} = \min\{p_{1,i}, p_{2,i+1}\} > \min\{p_{2,i}, p_{1,i+1}\}$

$$T: \sum_{k=1}^{i-1} p_{1,k} - \sum_{k=1}^{i-1} p_{2,k} + p_{1,i+1} + p_{1,i} - p_{2,i+1}$$

$$= \sum_{k=1}^{i-1} p_{1,k} - \sum_{k=1}^{i-1} p_{2,k} + p_{1,i} + p_{1,i+1} - p_{2,i+1}$$

Comparing schedules S and T – Case 1

$$S: \sum_{k=1}^{i-1} p_{1,k} - \sum_{k=1}^{i-1} p_{2,k} + p_{1,i} + \max\{0, p_{1,i+1} - p_{2,i}\}$$

$$\Rightarrow T - S: p_{1,i+1} - p_{2,i+1} - \max\{0, p_{1,i+1} - p_{2,i}\}$$

$$< \underbrace{p_{1,i+1} - \min\{p_{2,i}, p_{1,i+1}\}}_{\substack{=0 \text{ if } p_{2,i} \geq p_{1,i+1} \\ = p_{1,i+1} - p_{2,i}, \text{ otherwise}}} - \underbrace{\max\{0, p_{1,i+1} - p_{2,i}\}}_{\substack{=0, \text{ if } p_{2,i} \geq p_{1,i+1} \\ = p_{1,i+1} - p_{2,i}, \text{ otherwise}}} = 0$$

Comparing schedules S and T

$$T: \sum_{k=1}^{i-1} p_{1,k} - \sum_{k=1}^{i-1} p_{2,k} + p_{1,i+1} + \max\{p_{1,i} - p_{2,i+1}, 0\}$$

$$S: \sum_{k=1}^{i-1} p_{1,k} - \sum_{k=1}^{i-1} p_{2,k} + p_{1,i} + \max\{0, p_{1,i+1} - p_{2,i}\}$$

We know: $\min\{p_{1,i}, p_{2,i+1}\} > \min\{p_{2,i}, p_{1,i+1}\}$

Comparing schedules S and T – Case 2

Case 2: $p_{1,i} = \min\{p_{1,i}, p_{2,i+1}\} > \min\{p_{2,i}, p_{1,i+1}\}$

$$T: \sum_{k=1}^{i-1} p_{1,k} - \sum_{k=1}^{i-1} p_{2,k} + p_{1,i+1}$$

$$S: \sum_{k=1}^{i-1} p_{1,k} - \sum_{k=1}^{i-1} p_{2,k} + p_{1,i} + \max\{0, p_{1,i+1} - p_{2,i}\}$$

$$\Rightarrow T - S: p_{1,i+1} - p_{1,i} - \max\{0, p_{1,i+1} - p_{2,i}\}$$

$$< \underbrace{p_{1,i+1} - \min\{p_{2,i}, p_{1,i+1}\}}_{\substack{=0 \text{ if } p_{2,i} \geq p_{1,i+1} \\ = p_{1,i+1} - p_{2,i}, \text{ otherwise}}} - \underbrace{\max\{0, p_{1,i+1} - p_{2,i}\}}_{\substack{=0, \text{ if } p_{2,i} \geq p_{1,i+1} \\ = p_{1,i+1} - p_{2,i}, \text{ otherwise}}} = 0$$

Conclusion

- T is not worse than S in both cases
- As a consequence, each optimal schedule can be transformed into a Johnson schedule without losing its optimality
- This completes the proof

Example

- Given: 2 Machines A and B, and 5 jobs to be processed
- Processing times

Machine	Jobs				
	1	2	3	4	5
A	20	11	13	5	17
B	15	27	8	27	13

Steps of Johnson's algorithm

1. Minimum is 4 on A
   Consequence: First possible position
   (4,-,-,-,-)
2. Minimum is 3 on B
   Consequence: Last possible position
   (4,-,-,-,3)
3. Minimum is 2 on A
   Consequence: First possible position
   (4,2,-,-,3)
4. Minimum is 5 on B
   Consequence: Last possible position
   (4,2,-,5,3)
5. Complete optimal schedule is **(4,2,1,5,3)**

4.5.2 The multiple-stage case

- Now, we consider the general case M>2
- Unfortunately, it was shown that these problems are NP-hard
- Therefore, we introduce a simple heuristic approach in the following

Palmer's heuristic

- The guideline suggested by Palmer as a very first heuristic for sequencing M-staged flow-shop systems is as follows
- Give priority to jobs with the strongest tendency to **progress from short times to long times in the sequence of operations**
- In detail, Palmer proposes the following priority calculation to measure this attribute in each job

Palmer's heuristic

$$\forall j \in \{1,\ldots,N\}: s_j = \sum_{k=1}^{M} (M - 2\cdot(k-1) - 1)\cdot t_{j,M-k+1}$$

$$\Rightarrow M = 2:$$
$$s_j = (2-2\cdot 0 - 1)\cdot t_{j,2} + (2-2\cdot 1-1)\cdot t_{j,1} = t_{j,2} - t_{j,1}$$
$$\Rightarrow M = 3:$$
$$s_j = (3-2\cdot 0-1)\cdot t_{j,3} + (3-2\cdot 1-1)\cdot t_{j,2} + (3-2\cdot 2-1)\cdot t_{j,1}$$
$$= 2\cdot t_{j,3} + 0\cdot t_{j,2} - 2\cdot t_{j,1}$$
$$\Rightarrow M = 4:$$
$$s_j = (4-2\cdot 0-1)\cdot t_{j,4} + (4-2\cdot 1-1)\cdot t_{j,3} + (4-2\cdot 2-1)\cdot t_{j,2} + (4-2\cdot 3-1)\cdot t_{j,1}$$
$$= 3\cdot t_{j,4} + t_{j,3} - t_{j,2} - 3\cdot t_{j,1}$$

Solution

- The jobs are scheduled in sequence of non-increasing priority
- Generates only solutions with an equal sequence at all stages

Example

Processing time of job j on machine	Jobs			
	1	**2**	**3**	**4**
$t_{j,1}$	3	11	7	10
$t_{j,2}$	4	1	9	12
$t_{j,3}$	10	5	13	2

Priorities

Processing time of job j on machine	Jobs			
	1	2	3	4
$t_{j,1}$ -2	3 -6	11 -22	7 -14	10 -20
$t_{j,2}$ 0	4 0	1 0	9 0	12 0
$t_{j,3}$ 2	10 20	5 10	13 26	2 4
Priority	14	-12	12	-16

Solution

Is 1 – 3 – 2 – 4

CDS Heuristic

- CDS="Cambel Dudek Smith", the authors of the respective paper
- Extension of the Johnson algorithm for multiple-stage cases
- Considers only solutions with equal sequences at all stages
- Note that it starts from at least four stages
- Generates artificial 2-staged problems out of the general constellation and solves them optimally by the application of the Johnson algorithm
- For M=2 the CDS procedure becomes the Johnson algorithm generating an optimal solution
- Otherwise, the procedure generates M-1 iterations representing an additional two-staged flow-shop problem
- Can be used for the minimization of cycle time or total lead time

CDS procedure

1. Establish NxM-matrix of processing times $t_{j,i}$, where $t_{j,i}$ is the processing time of j-th job on machine i
2. Establish number of auxiliary n-job, 2-machine problems, p, to be calculated, where p≤M-1
3. Set k=1 for first auxiliary problem
4. Compute the processing time for all jobs j=1,…,N on the two machines in the k-th auxiliary problem:

$$\boxed{\theta_{j,1}^{k} = \sum_{i=1}^{k} t_{j,i}}$$

CDS Procedure

5. Compute the processing time for all jobs j=1,…,N on the two machines in the k-th auxiliary problem:

$$\theta^k_{j,2} = \sum_{i=M-k+1}^{M} t_{j,i}$$

6. Solve the problem with the Johnson algorithm
7. Check if k<p. If so, set k=k+1, go to step 3; Otherwise proceed with step 8
8. Use the original problem to compute the objective value of all p generated solutions
9. Select best result as the output of the procedure

Example

- Given: 4 Machines A, B, C and D, as well as 5 jobs to be processed
- Processing times

Machine	Jobs				
	1	2	3	4	5
A	3	6	10	4	7
B	12	4	1	1	9
C	1	2	6	7	4
D	6	1	2	8	1

First iteration

Machine	Jobs				
	1	2	3	4	5
1	3	6	10	4	7
2	6	1	2	8	1

Steps of Johnson's algorithm

1. Minimum is 2 on machine 2
   Consequence: Last possible position
   (-,-,-,-,2)
2. Minimum is 5 on machine 2
   Consequence: Last possible position
   (-,-,-,5,2)
3. Minimum is 3 on machine 2
   Consequence: Last possible position
   (-,-,3,5,2)
4. Minimum is 1 on machine 1
   Consequence: Last possible position
   (1,-,3,5,2)
5. Complete optimal schedule is **(1,4,3,5,2)**

Objective function value

Processing	Job 1		Job 4		Job 3		Job 5		Job 2	
	S	E	S	E	S	E	S	E	S	E
Machine A	0	3	3	7	7	17	17	24	24	30
Machine B	3	15	15	16	17	18	24	33	33	37
Machine C	15	16	16	23	23	29	33	37	37	39
Machine D	16	22	23	31	31	33	37	38	39	**40**

Objective function value

Processing	Job 4		Job 1		Job 3		Job 5		Job 2	
	S	E	S	E	S	E	S	E	S	E
Machine A	0	4	4	7	7	17	17	24	24	30
Machine B	4	5	7	19	19	20	24	33	33	37
Machine C	5	12	19	20	20	26	33	37	37	39
Machine D	12	20	20	26	26	28	37	38	39	**40**

Second iteration

Machine	Jobs				
	1	2	3	4	5
1	15	10	11	5	16
2	7	3	8	15	5

Steps of Johnson's algorithm

1. Minimum is 2 on machine 2
   Consequence: Last possible position
   (-,-,-,-,2)
2. Minimum is 5 on machine 2
   Consequence: Last possible position
   (-,-,-,5,2)
3. Minimum is 4 on machine 1
   Consequence: Last possible position
   (4,-,-,5,2)
4. Minimum is 1 on machine 2
   Consequence: Last possible position
   (4,-,1,5,2)
4. Complete optimal schedule is **(4,3,1,5,2)**

Objective function value

Processing	Job 4		Job 3		Job 1		Job 5		Job 2	
	S	E	S	E	S	E	S	E	S	E
Machine A	0	4	4	14	14	17	17	24	24	30
Machine B	4	5	7	8	17	29	29	38	38	42
Machine C	5	12	12	18	29	30	38	42	42	44
Machine D	12	20	20	22	30	36	42	43	44	**45**

Third iteration

Machine	Jobs				
	1	2	3	4	5
1	16	12	17	12	20
2	19	7	9	16	14

Steps of Johnson's algorithm

1. Minimum is 2 on machine 2
   Consequence: Last possible position
   (-,-,-,-,2)
2. Minimum is 3 on machine 2
   Consequence: Last possible position
   (-,-,-,3,2)
3. Minimum is 4 on machine 1
   Consequence: Last possible position
   (4,-,-,3,2)
4. Minimum is 5 on machine 2
   Consequence: Last possible position
   (4,-,5,3,2)
4. Complete optimal schedule is **(4,1,5,3,2)**

Objective function value

Processing	Job 4		Job 1		Job 5		Job 3		Job 2	
	S	E	S	E	S	E	S	E	S	E
Machine A	0	4	4	7	7	14	14	24	24	30
Machine B	4	5	7	19	19	28	28	29	30	34
Machine C	5	12	19	20	28	32	32	38	38	40
Machine D	12	20	20	26	32	33	38	40	40	**41**

Output

- Best found solution was (4,1,3,5,2)
- Objective value: 40

Applying Palmer's procedure

Machine	Jobs				
	1	2	3	4	5
A **-3**	3 -9	6 -18	10 -30	4 -12	7 -21
B **-1**	12 -12	4 -4	1 -1	1 -1	9 -9
C **1**	1 1	2 2	6 6	7 7	4 4
D **3**	6 18	1 3	2 6	8 24	1 3
Priority	-2	-17	-19	18	-23

Result of Palmer's procedure

- 4 – 1 – 2 – 3 – 5

Objective function value

Processing	Job 4		Job 1		Job 2		Job 3		Job 5	
	S	E	S	E	S	E	S	E	S	E
Machine A	0	4	4	7	7	13	13	23	23	30
Machine B	4	5	7	19	19	23	23	24	30	39
Machine C	5	12	19	20	23	25	25	31	39	43
Machine D	12	20	20	26	26	27	31	33	43	**44**

Result of Palmer's procedure

Distribution of Error in Terms of Percent Deviation of Algorithm Best Sequence Time from Optimal Sequence Time

n	m	Sample Size	Numbers of Problems with Percent Deviation of 0%	0% < to ≤ 5%	>5%	Range (%)	Average Error (90)	
3	3	20	17	3	0	0–2.9	0.19	
4	3	20	15	3	2	0–22.6	1.80	
5	3	20	13	6	1	0–10.4	1.13	1.65
6	3	20	12	2	6	0–15.9	3.57	
7	3	20	13	5	2	0–8.3	1.17	
8	3	20	12	5	3	0–14.7	2.03	
3	5	20	19	1	0	0–1.4	0.07	
4	5	20	9	7	4	0–41.3	3.95	
5	5	20	11	5	4	0–9.2	2.02	3.77
6	5	20	4	7	9	0–23.2	5.24	
7	5	20	2	9	9	0–14.3	5.18	
8	5	20	3	7	10	0–25.4	6.18	
3	7	20	20	0	0	0	0	
4	7	20	14	5	1	0–5.3	0.68	
5	7	20	7	9	4	0–10.0	2.27	2.13
6	7	20	3	9	8	0–11.4	3.64	
7	7	20	2	12	6	0–12.6	4.07	
Total		340	176	95	69	Average	2.54	

Source: Palmer (1964)

Comparison with Palmer heuristic

Error Comparison on Problems with Known Optimal Sequence Times

Problem Size n	m	No. Problems	Campbell-Dudek No. Optimals	Largest Error (%)	Average Error (%)		Palmer No. Optimals	Largest Error (%)	Average Error (%)	
3	4	20	17	8.1	.61		11	17.5	2.68	
4	4	20	15	10.1	1.74	1.54	6	19.2	6.08	4.58
5	4	20	10	13.8	2.56		4	15.6	4.95	
6	4	20	12	11.5	1.24		2	15.1	4.61	
3	6	20	18	1.4	.12		14	18.7	2.77	
4	6	20	15	4.1	.56	1.22	8	13.9	3.13	4.58
5	6	20	11	6.9	2.01		2	20.2	5.90	
6	6	20	6	9.0	2.18		3	18.2	6.52	
Totals		160	104		1.38		50		4.58	

Comparison on Problems with Unknown Optimal Sequence Times

n	m	Sequence Time C-D	Palmer	% Improvement
20	20	2452	2712	10.60
20	20	2496	2542	1.84
20	20	2390	2382	−0.34
20	20	2422	2484	2.56
40	30	4458	4574	2.60
40	30	4597	4634	0.80
40	30	4496	4600	2.31
40	30	4475	4665	4.25
60	30	5747	5841	1.64
60	30	5849	5997	2.53

Computational time

Computer Computation Time

Number of Jobs	Average Computation Time (min) C-D	Palmer
8	.055	.029
10	.067	.037
20	.195	.100
40	.752	.223
60	1.806	.347

Pros and Cons

- **Pros**
  - CDS/Palmer are fast to compute
  - CDS generates quite good solutions
- **Cons**
  - Poor results in comparison to elaborated meta strategies
  - Used for finding an initial solution but not for the final result

Some additional references to Section 4

- Adams, J.; Balas, E.; Zawack, D.: The Shifting Bottleneck Procedure for Job Shop Scheduling. Management Science, Vol.34, No.3, 1988.
- Baker, K.R.: Introduction to Sequencing and Scheduling. John Wiley&Sons, New York et al., 1974. (**ISBN-10:** 0-4710-4555-1)
- Brucker, P.: Scheduling Algorithms. Springer, Berlin et al., 5th edition, 2007. (**ISBN-10:** 3-5406-9515-X)
- Carlier, J.: The one-machine sequencing problem. European Journal of Operational Research, 11, pp.42-47, 1982.
- Campbell, H.G.; Dudek, R.A.; Smith, M.L.: A Heuristic Algorithm for the n Job m Machine Sequencing Problem. Management Science, Vol.16, No.10, 1970.
- Nowicki, E.; Smutnicki, C.: A Fast Taboo Search Algorithm for the Job Shop Problem. Management Science Vol.42, No.6, 1996.
- Palmer, D.S.: Sequencing Jobs Through a Multi-Stage Process in the Minimum Total Time – A Quick Method of Obtaining a Near Optimum. Operational Research Quarterly, Vol.16, No.1, 1964.
- Pinedo, M.L.: Scheduling: Theory, Algorithms and Systems. 4th edition, Prentice Hall, New Jersey, 2012. (**ISBN-10:** 1-4614-1986-7)