Combinatorial Optimization Problems

Scheduling Problems

Gábor Galambos

Heidelberg
2005 februar

The problem: There are m machines that are used to process n jobs. A schedule specifies, for each machine i ($i=1,2,\ldots,m$) and each job j ($j=1,2,\ldots,n$) one ore more time intervals throughout which processing is performed on j by i.

A schedule is feasible if
- there is no overlapping of time intervals corresponding to the same job,
- there is no overlapping of time intervals corresponding to the same machine,
- it is satisfies various requirements related to the specific problem type.

An operation refers to a specified period of processing by some machine type.

We assume that all machines become available to process jobs at time zero.

A problem type is specified by

- the machine environment
- the job characteristic
- an optimality criterion.

The machine environment I.

<u>Single-stage production systems</u>: each job requires one operation.
- single machine
- m machines operating parallel

identical parallel machines: each processing time is independent of the machine performing the job.

uniformly parallel machines: the machines operate at different speeds but otherwise they are identical.

unrelated parallel machines: the processing time of a job depends on the machine assignement.

The machine environment II.

Multi-stage production systems: there are s stages, each having a different function. In a

flow shop the processing of each job goes through the stages $1,2,\dots,s$ in that order,

open shop like the flow shop, but the routing that specifies the sequence of stages through which a job must pass, can differ between jobs and forms part of the decision process,

job shop each job has a prescribed routing through the stages, and the routing may differ from job to job.

Job characteristics I.

A job may be characterized by its

- Processing requirements
- Availability for processing
- Precedence constraints
- Interruption conditions

Processing requirements

For the job j the processing time is denoted in case of

- single machine and identical parallel machines by p_j,
- uniform parallel machines on machine i may be expressed as p_j/s_i, where s_i is the speed of machine i,
- unrelated parallel machines

  for flow shop and open shop p_{ij} is the processing time on machine/stage i,

  for job shop p_{ij} denotes the processing time of the ith operation (which is not necesseraly performed at stage i).

We can assume that all p_j and p_{ij} are integers.

We will denote by p_{max} the maximum value of all p_i or p_{ij}.

Availability for processing

The availability of each job j may be restricted by its

- integer release date r_j that defines when it becomes available for processing,
- integer deadline d_j that specifies the time by which it must be completed. (It is called sometimes as duedate).

Precedence constraints, and interruption

If job j has precedence over job k, then cannot start its processing until j is completed.

Precedence constraints are specified by a directed acyclic precedence graph G with vertices $1,2,\ldots,n$: there is a directed path from vertex j to vertex k if and only if job j has precedence over job k.

If the processing of any operation may be interrupted and resumed at a later time on the same or on a different machine then the model allows the preemption.

Optimality criteria I.

For each job j, an integer due date d_j and a positive integer weight w_j may be specified. For a given schedule σ, we can compute for job j:

completion time	$C_j(\sigma)$	C_j	
flow time	$F_j(\sigma)$	F_j	$= C_j(\sigma) - r_j$
lateness	$L_j(\sigma)$	L_j	$= C_j(\sigma) - d_j$
earliness	$E_j(\sigma)$	E_j	$= max\{d_j - C_j(\sigma), 0\}$
tardiness	$T_j(\sigma)$	$T_j(\sigma)$	$= max\{C_j(\sigma) - d_j, 0\}$
unit penalty	$U_j(\sigma)$	U_j	$= \begin{cases} 1 \text{ if } C_j(\sigma) > 0 \\ 0 \text{ otherwise.} \end{cases}$
cost	$f_j(\sigma)$	f_j	$= f(C_j(\sigma))$
allowance	$a_j(\sigma)$	a_j	$= d_j - r_j$

Model classification

Off-line model: the scheduler has full information of the problem instance, such as total number of jobs, their released dates and processing times, before the process of scheduling actually starts.

On-line model: information about the problem instance is made available to the scheduler job by job during the course of scheduling.

In nearly on-line scheduling the released date of next job is always known to the scheduler.

In each model we assume that the scheduler´s decision to assign and schedule a job or operation is irrevocable.

On-line model classification I

The classes are different according to the way of information on job characteristics is released to the scheduler. We distinguish:

- Scheduling over list, where the scheduler is confronted with the jobs one-by-one as they appear in the list. The existence of a job is not known until all ist predecessors have already been scheduled.
- Scheduling over time, where all jobs arrive at their release dates. The jobs are scheduled with the passage of time and, at any time, the scheduler only has knowledge of those jobs that have already arrived.

On-line model classification II

Both above model (scheduling over list, scheduling over time) supposes that once a job is known to the scheduler, its processing requirement is also known. So we call these models clairvoyant.

In case of non-clairvoyant model the processing requirem-ent of a job is unknown until ist processing is completed.

The optimality criteria involve

A. The minimization of

maximum completion time (makespan)	C_{max}	$max_j\ C_j$
maximum lateness	L_{max}	$max_j\ L_j$
maximum cost	f_{max}	$max_j\ f_j$
maximum tardiness	T_{max}	$max_j T_j$
maximum flow time	F_{max}	$max_j F_j$
maximum earliness	E_{max}	$max_j\ E_j$
maximum allowance	a_{max}	$max_j\ a_j$
maximum waiting time	W_{max}	$max_{ik}\ W_{ik}$

The optimality criteria involve

B. The minimization of

total (weighted) completion time	$\Sigma_j(w_j)C_j$	
total (weighted) flow time	$\Sigma_j(w_j)F_j$	
total (weigted) earliness	$\Sigma_j(w_j)E_j$	
total (weighted) tardiness	$\Sigma_j(w_j)T_j$	
total (weighted) allowance	$\Sigma_j(w_j)a_j$	
total (weighted) waiting time	$\Sigma_j(w_j)W_j$	
(weighted) number of late jobs	$\Sigma_j(w_j)U_j$	
total cost	$\Sigma_j f_j$	

Usually, we call an optimality criteria $\mathcal{R}$ as regular, if it is non-deacreasing in the completion times.

If $\mathcal{R}$ is regular, then if

$$C_1 \leq C_1', C_2 \leq C_2', \text{ and } \quad C_n \leq C_n',$$

then

$$\mathcal{R}(C_1, C_2, \ldots, C_n) \leq \mathcal{R}(C_1', C_2', \ldots, C_n')$$

Theorem: C_{max}, F_{max}, L_{max}, T_{max} are regular objective functions.

Three-field representation
(Graham, Lawler, Lenstra, Rinnooy Kann)

We will use a three-field descriptor:

$\alpha \mid \beta \mid \gamma$

Machine enviroment

Job characteristic

Optimality criteria

Let ○ denote the empty symbol.

Let $\alpha = \alpha_1\alpha_2\alpha_3$ where

$\alpha_1 \in \{○, P, Q, R, F, O, J\}$

- ○ single machine
- *P* identical parallel machine
- *Q* uniform parallel machine
- *R* unrelated parallel machine
- *O* open shop
- *F* flow shop
- *J* job shop

$\alpha_2 \in \{○, m, s\}$

- ○ single machine
- *m* the number of machine is *m*
- *s* the number of stages is *s*

$\alpha_3 \in \{○, (Pm), (Pm_1, \ldots, Pm_s), (P)\}$

- ○ single stage or several stages each with a single machine
- *(Pm)* multi-stage with *m* identical parallel machines at each stage
- $(Pm_1, \ldots, Pm_s)$ multi-stage with m_k identical parallel machines at stage *k*
- *(P)* multi-stage with an arbitrary number of identical parallel machines at each stage

$\beta \subseteq \{\beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6\}$ where

$\beta_1 \in \{○$, *on-line-list, on-line, on-line-list-nclv, on-line-nclv*$\}$

$\beta_2 \in \{○, r_j\}$ indicates whether jobs have release time.

$\beta_3 \in \{○, d_j\}$ indicates whether jobs have deadlines.

$\beta_4 \in \{○, pmtn\}$ indicates whether jobs may be preempted.

$\beta_5 \in \{○, prec\}$ indicates whether jobs have precedence constraints.

$\beta_6 \in \{○, \underline{p_j = 1, p_{ij} = 1}\}$ indicates whether jobs have unit processing time.

$\gamma \in \{\underline{C_{max}, L_{max}, E_{max}, T_{max}, f_{max}, \ldots}\}$

Examples

$1 \mid r_j, prec \mid \sum_j w_j C_j$ is the problem of scheduling jobs with release dates and precedence constraints on a single machine to minimize the total weighted completion time.

$R \mid pmtn \mid L_{max}$ is the problem of preemptively scheduling jobs on an arbitrary number of unrelated parallel machines to minimize the maximum lateness.

$O3 \mid p_{ij} = 1 \mid \sum_j Uj$ is the problem of scheduling jobs in a three-machine open shop to minimize the number of late jobs, where the processing time of each operation is one unit.

The Coupled-Tasks Problem(CTP):

We are given n jobs.Each of them consisting of two distinct operations. The sequence of operations are fixed and also a fixed length of delay-time passes between the two parts.

The i-te job is denoted by a triple (a_i,L_i,b_i) where
- a_i = the processing time of the first task,
- L_i = the delay time between the tasks, and
- b_i = the processing time of the second task.

The aim is to schedule the n coupled-tasks on one machine in such a way that
- no jobs are overlaped
- the makespan (C_{max}) has to be minimized.
- no preemption is allowed.

$$1|Coup\text{-}Task|C_{max}$$

Theorem: A schedule which is optimal with respect to L_{max} is also optimal with respect to T_{max}.

Proof.

$$T_{max} = max\{max\{L_1,0\}, max\{L_2,0\},\ldots,max\{L_n,0\}, \}$$
$$= max\{L_1,L_2,\ldots,L_n,0\}$$
$$= max\{L_{max},0\}$$

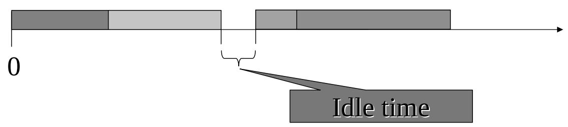
□

Methodologies for solving a scheduling problem

1. Examine the complexity of the problem
   - P → $1\,|\,|\,\sum w_jC_j$
   - NP → J $|\,|\,C_{max}$
2. In case of P we use
   - Greedy algorithms
     - SPT , $SWPT$ rule → $1\,|\,|\,\sum w_jC_j$
     - EDD rule → $1\,|\,rj\,|\,L_{max}$
   - Enumerative algorithms
     - Dynamic Programing → $1\,|\,|\,\sum f_j$
     - Branch and Bound → $1\,|\,r_j\,|\,L_{max}$
3. In case of NP we use
   - Local Search → $P\,|\,|\,C_{max}$
   - Approximation algorithms → $P\,|\,|\,C_{max}$

Single Machine Processing

Example: Processing of jobs through a small non-time-sharing computer.



Theorem: for an ○ | β_1 | $\mathcal{R}$ problem, where β_1 is arbitrary processing condition and $\mathcal{R}$ is also a regular optimality criterion, there exists an optimal schedule in which there is no inserted idle time, i.e. the machine starts processing at $t = 0$ and continues without rest until $t = C_{max}$.

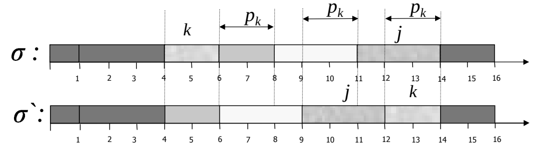
Theorem: for an ○ | β_1 | $\mathcal{R}$ problem, where β_1 is arbitrary processing condition and $\mathcal{R}$ is also a regular optimality criterion, no improvement may be gained in the optimal schedule by allowing preemption.

Consequence: In case of regular optimality criterion we need to consider the permutatition schedules, i.e. to find a permutation of the jobs such that when they are sequenced in that order, the value of $\mathcal{R}$ is minimised.

Shortest Processing Time Scheduling

Theorem (Jackson, 1955): The problem $1 \mid \mid \underline{L_{max}}$ can be solved in $O(nlogn)$ time by sequencing jobs in non-decreasing order of their due dates (*EDD* rule).

Proof. Consider any optial sequence σ, and suppose that some job k is sequenced before another job j, where $d_k > d_j$.



Observation 1: all jobs sequenced before k and after j in σ have the <u>same completion</u> time in both sequences.

Observation 2: job j together with all jobs sequenced between k and j in σ are completed <u>p_k units earlier</u> in σ`.

$$L_k(\sigma`) = \underline{C_k(\sigma`) - d_k} = \underline{C_j(\sigma) - d_k} < C_j(\sigma) - d_j = L_j(\sigma).$$

So

$L_{max}(\sigma`) \leq L_{max}(\sigma)$ ⟹ $\sigma$` is also optimal.

Repetition of this job reinsertion argument yields an optimal sequence in which jobs appear in EDD order. □

Exercise (Smith, 1956) : Let us prove that for the problem $1||wjCj$ an optimal solution is obtained in $O(nlogn)$ time by sequencing jobs in non-decreasing order of p_j/w_j. (*SWPT* rule)

Hints: use an adjacent interchange procedure for those of pairs where $p_k/w_k > p_j/w_j$.

Lemma: In an optimal schedule the first k jobs ($k=1,2,\ldots,n$) also formed an optimal schedule for the reduced problem based on just these k jobs alone.

Proof: Let ($j_{i_1},j_{i_2},\ldots,j_{i_n}$) be an optimal schedule. Then for any k, $k=1,\ldots,n$, we may decompose the objective function:

$$min \sum_{j=1}^{n} T_j = min \sum_{j=1}^{k} T_j + min \sum_{j=k+1}^{n} T_j$$

Let us now consider the set of jobs $\{j_{i_1},j_{i_2},\ldots,j_{i_k}\}$, and produce the optimal schedule. If we can improve upon the sequence ($j_{i_1},j_{i_2},\ldots,j_{i_k}$), then we would able to reduce the first term above.

⇓

We can construct a new sequence for the full problem by using the the improved scheduling for the first k jobs, and leaving the remaining (n-k) jobs in their original order. □

The Total Weighted Tardiness

Theorem (Held, Karp, 1962): The problem $1 \mid\mid \sum T_j$ can be solved by dynamic programming technique.

To prove the theorem we need the following Lemma.

- Let the period k be the situation when we consider the sets of k elements.
- The state set S_k in the k-te period contains all schedules belonging to the permutations of the t jobs in hand.
- During the decision procedure in the k-te period we choose a permutation with minimum property according to the objective function.

S_k = {the set of all permutations of jobs $j_1,j_2,\ldots,j_k$}

The decision variables are:

in S_1: $X_1=\{ j_1, j_2,\ldots, j_n\}$

in S_2: $X_1=\{\{j_1, j_2\},\ldots,\{j_{n-1}, j_n\}\}$

…

in S_{n-1}: $X_1=\{\{j_1, j_2,\ldots,j_{n-1}\},\ldots, \{j_2, j_3,\ldots,j_n\}\}$

The states are the values of the objective function:

$$z_k = z_k(S_k) = \min_{S_k} \sum_{j_i \in S_k} T_{j_i} = \min_{\pi \in S_k} \sum_{j_i \in \pi} max\,(C_{j_i} - d_{j_i})$$

and so

$$z_k = \min_{j_i \in S_k} \{z_{k-1}(S_k - \{j_i\}) + T_{j_i}\}$$

Example: Solve the problem $1 \mid\mid \sum T_j$ if

	j_1	j_2	j_3	j_4
Processing time (p_i)	8	6	10	7
due date (d_i)	14	9	16	16

Let us calculate the dynamic programming procedure.

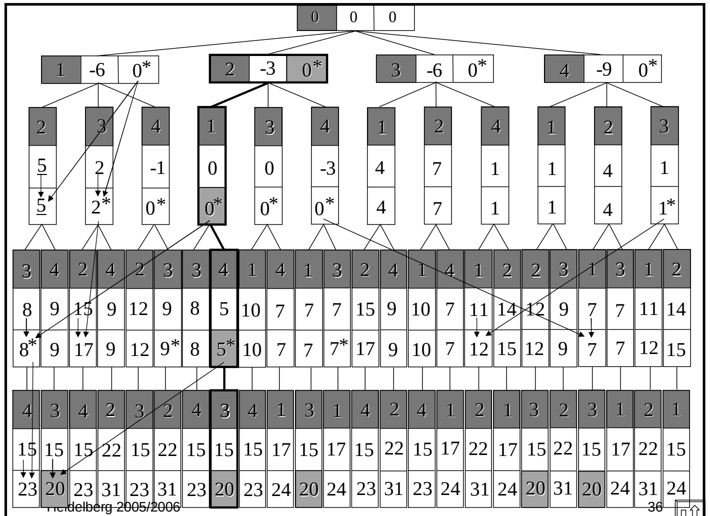
The optimal schedule is: { j_3, j_4, j_1, j_2}

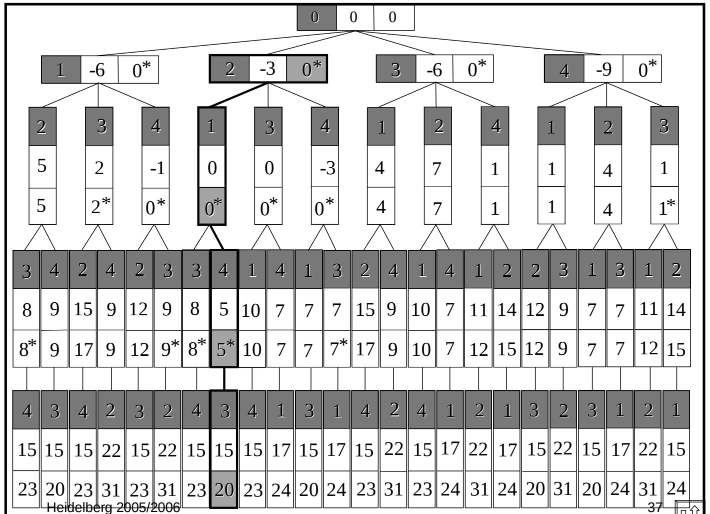
and

$z = min\ max \sum T_{j_i} = min \sum max\,(C_{j_i} - d_{j_i}) = 20$

Computational complexity of the dynamic programming

n	Number of operation required	
	Complete enumeration	Dynamic programming
	6n(n!)+3(n!-1)	$6n2^{n-1}+3(2^n-1)$
4	*647*	*237*
10	2.286×10^{8}	*33789*
20	2.992×10^{20}	6.396×10^{7}
40	1.983×10^{50}	1.352×10^{14}





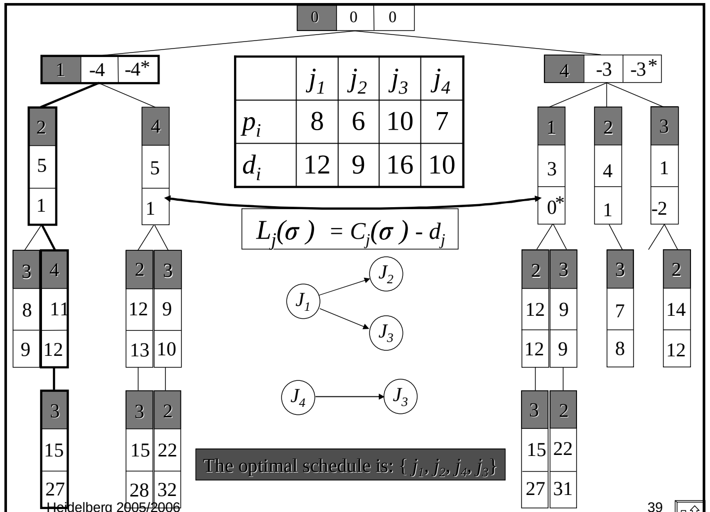
Dynamic Programming to precedence constraints

Example: Solve the problem $1 \mid\mid \sum L_j$ if

	j_1	j_2	j_3	j_4
Processing time (p_i)	8	6	10	7
due date (d_i)	12	9	16	10

and the precedence constrains are:

$L_j(\sigma) = C_j(\sigma) - d_j$



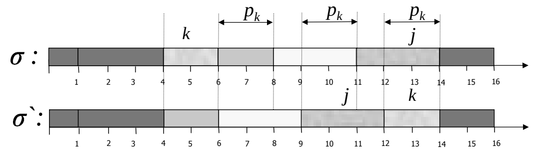
	j_1	j_2	j_3	j_4
p_i	8	6	10	7
d_i	12	9	16	10

$L_j(\sigma) = C_j(\sigma) - d_j$

The optimal schedule is: { j_1, j_2, j_4, j_3}

Theorem (Dominance condition): in an $1\|\sum T_j$ problem if two jobs J_i and J_k are such that $p_i \leq p_j$ and $d_i \leq d_j$ then exists an optimal schedule in which J_i precedes J_k.

Proof: Let us suppose that we have an optimal schedule σ where the conditions hold and J_k precedes J_i.



$L_k(\sigma`) = \underline{C_k(\sigma`) - d_k}$

$L_k(\sigma`) = \underline{C_k(\sigma`) - d_k} = \underline{C_j(\sigma) - d_k} < C_j(\sigma) - d_j = L_j(\sigma).$