Scheduling
Single Machine Scheduling

Tim Nieberg

Single machine models

Observation:

- for non-preemptive problems and regular objectives, a sequence in which the jobs are processed is sufficient to describe a solution

Dispatching (priority) rules

- static rules - not time dependent
  e.g. shortest processing time first, earliest due date first
- dynamic rules - time dependent
  e.g. minimum slack first (slack$= d_j - p_j - t$; t current time)
- for some problems dispatching rules lead to optimal solutions

Single machine models: $1||\sum w_j C_j$

Given:

- n jobs with processing times $p_1, \ldots, p_n$ and weights $w_1, \ldots, w_n$

Consider case: $w_1 = \ldots = w_n (= 1)$:

Single machine models: $1||\sum w_j C_j$

Given:

- n jobs with processing times $p_1, \ldots, p_n$ and weights $w_1, \ldots, w_n$

Consider special case: $w_1 = \ldots = w_n (= 1)$:

- SPT-rule: shortest processing time first
- Theorem: SPT is optimal for $1||\sum C_j$
  Proof: by an exchange argument (on board)
- Complexity: $O(n \log(n))$

Single machine models: $1||\sum w_j C_j$

General case

- WSPT-rule: weighted shortest processing time first, i.e. sort jobs by increasing p_j/w_j-values
- Theorem: WSPT is optimal for $1||\sum w_j C_j$
  Proof: by an exchange argument (exercise)
- Complexity: $O(n\log(n))$

Further results:

- $1|tree|\sum w_j C_j$ can be solved by in polynomial time ($O(n\log(n))$)
  (see [Brucker 2004])
- $1|prec|\sum C_j$ is NP-hard in the strong sense
  (see [Brucker 2004])

Single machine models: $1|prec|f_{max}$

Given:

- n jobs with processing times $p_1, \ldots, p_n$
- precedence constraints between the jobs
- regular functions $f_1, \ldots, f_n$
- objective criterion $f_{max} = \max\{f_1(C_1), \ldots, f_n(C_n)\}$

Observation:

- completion time of last job $= \sum p_j$

Single machine models: $1|prec|f_{max}$

Given:

- n jobs with processing times $p_1, \dots, p_n$
- precedence constraints between the jobs
- regular functions $f_1, \dots, f_n$
- objective criterion $f_{max} = \max\{f_1(C_1), \dots, f_n(C_n)\}$

Observation:

- completion time of last job $= \sum p_j$

Method

- plan backwards from $\sum p_j$ to 0
- from all available jobs (jobs from which all successors have already been scheduled), schedule job which is 'cheapest' on that position

Single machine models: $1|prec|f_{max}$

S	set of already scheduled jobs (initial: $S = \emptyset$)
J	set of all jobs, which successors have been scheduled (initial: all job
t	time where next job will be completed (initial: $t = \sum p_j$)

Algorithm $1|prec|f_{max}$ (Lawler's Algorithm)

REPEAT

select $j \in J$ such that $f_j(t) = \min_{k \in J} f_k(t)$;
schedule j such that it completes at t;
add j to S, delete j from J and update J;
$t := t - p_j$;

UNTIL $J = \emptyset$.

Single machine models: $1|prec|f_{max}$

- Theorem: Algorithm $1|prec|f_{max}$ is optimal for $1|prec|f_{max}$
  Proof: on the board
- Complexity: $O(n^2)$

Single machine models: Maximum Lateness

Problem $1||L_{max}$:

- Earliest due date first (EDD) is optimal for $1||L_{max}$ (Jackson's EDD rule)
- Proof: special case of Lawler's algorithm

Single machine models: Maximum Lateness

Problem $1||L_{max}$:

- Earliest due date first (EDD) is optimal for $1||L_{max}$ (Jackson's EDD rule)
- Proof: special case of Lawler's algorithm

Problem $1|r_j|C_{max}$:

- $1|r_j|C_{max} \propto 1||L_{max}$
  - define $d_j := K - r_j$, with constant $K > \max r_j$
  - reversing the optimal schedule of this $1||L_{max}$-problem gives an optimal schedule for the $1|r_j|C_{max}$-problem

Single machine models: Maximum Lateness

<u>Problem $1|prec|L_{max}$</u>:

- if $d_j < d_k$ whenever $j \rightarrow k$, any EDD schedule respects the precedence constraints, i.e. in this case EDD is optimal
- defining $d_j := \min\{d_j, d_k - p_k\}$ if $j \rightarrow k$ does not increase L_{max} in any feasible schedule

Single machine models: Maximum Lateness

Problem $1|prec|L_{max}$:

- if $d_j < d_k$ whenever $j \to k$, any EDD schedule respects the precedence constraints, i.e. in this case EDD is optimal
- defining $d_j := \min\{d_j, d_k - p_k\}$ if $j \to k$ does not increase L_{max} in any feasible schedule

Algorithm $1|prec|L_{max}$

1. make due dates consistent:
   set $d_j = \min\{d_j, \min_{k|j \to k} d_k - p_k\}$
2. apply EDD rule with modified due dates

Remarks on Algorithm $1|prec|L_{max}$

- leads to an optimal solution
- Step 1 can be realized in $O(n^2)$
- problem $1|prec|L_{max}$ can be solved without knowledge of the processing times, whereas Lawler's Algorithm (which also solves this problem) in general needs this knowledge (Exercise),
- Problem $1|r_j, prec|C_{max} \propto 1|prec|L_{max}$

Problem $1|r_j|L_{max}$:

- problem $1|r_j|L_{max}$ is NP-hard
- Proof: by reduction from 3-PARTITION (on the board)

Single machine models: Maximum Lateness

Problem $1|pmtn, r_j|L_{max}$:

- preemptive EDD-rule: at each point in time, schedule an available job (job, which release date has passed) with earliest due date.
- preemptive EDD-rule leads to at most k preemptions ($k =$ number of distinct release dates)

Single machine models: Maximum Lateness

Problem $1|pmtn, r_j|L_{max}$:

- preemptive EDD-rule: at each point in time, schedule an available job (job, which release date has passed) with earliest due date.
- preemptive EDD-rule leads to at most k preemptions ($k =$ number of distinct release dates)
- preemptive EDD solves problem $1|pmtn, r_j|L_{max}$
- Proof (on the board) uses following results:
  - $L_{max} \geq r(S) + p(S) - d(S)$ for any $S \subset \{1, \ldots, n\}$, where $r(S) = \min_{j \in S} r_j,\ p(S) = \sum_{j \in S} p_j,\ d(S) = \max_{j \in S} d_j$
  - preemptive EDD leads to a schedule with $L_{max} = \max_{S \subset \{1,\ldots,n\}} r(S) + p(S) - d(S)$

Remarks on preemptive EDD-rule for $1|pmtn, r_j|L_{max}$:

- can be implemented in $O(n\log(n))$
- is an 'on-line' algorithm
- after modification of release and due-dates, preemptive EDD solves also $1|prec, pmtn, r_j|L_{max}$

Approximation algorithms for problem $1|r_j|L_{max}$:

- a polynomial algorithm A is called an α-approximation for problem P if for every instance I of P algorithm A yields an objective value $f_A(I)$ which is bounded by a factor α of the optimal value $f^*(I)$; i.e. $f_A(I) \leq \alpha f^*(I)$

Single machine models: Maximum Lateness

Approximation algorithms for problem $1|r_j|L_{max}$:

- a polynomial algorithm A is called an α-approximation for problem P if for every instance I of P algorithm A yields an objective value $f_A(I)$ which is bounded by a factor α of the optimal value $f^*(I)$; i.e. $f_A(I) \leq \alpha f^*(I)$
- for the objective L_{max}, α-approximation does not make sense since L_{max} may get negative
- for the objective T_{max}, an α-approximation with a constant α implies $\mathcal{P} = \mathcal{NP}$ (if $T_{max} = 0$ an α-approximation is optimal)

Single machine models: Maximum Lateness

The head-body-tail problem $(1|r_j, d_j < 0|L_{max})$

- n jobs
- job j: release date r_j (head), processing time p_j (body), delivery time q_j (tail)
- starting time $S_j \geq r_j$;
- completion time $C_j = S_j + p_j$
- delivered at $C_j + q_j$
- goal: minimize $\max_{j=1}^{n} C_j + q_j$

Single machine models: Maximum Lateness

The head-body-tail problem ($1|r_j, d_j < 0|L_{max}$), (cont.)

- define $d_j = -q_j$, i.e. the due dates get negative!
- result: $\min_{j=1}^{n} C_j + q_j = \min_{j=1}^{n} C_j - d_j = \min_{j=1}^{n} L_j = L_{max}$
- head-body-tail problem equivalent with $1|r_j|L_{max}$-problem with negative due dates
  Notation: $1|r_j, d_j < 0|L_{max}$
- an instance of the head-body-tail problem defined by n triples (r_j, p_j, q_j) is equivalent to an inverse instance defined by n triples (q_j, p_j, r_j)
- for the head-body-tail problem considering approximation algorithms makes sense

The head-body-tail problem $(1|r_j, d_j < 0|L_{max})$, (cont.)

- $L_{max} \geq r(S) + p(S) + q(S)$ for any $S \subset \{1, \ldots, n\}$, where

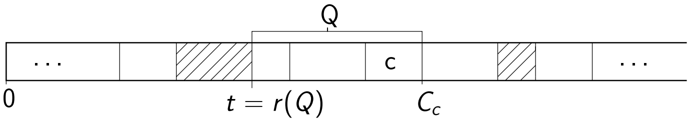
$$r(S) = \min_{j \in S} r_j, \;\; p(S) = \sum_{j \in S} p_j, \;\; q(S) = \min_{j \in S} q_j$$

(follows from $L_{max} \geq r(S) + p(S) - d(S)$)

Single machine models: Maximum Lateness

<u>Approximation ratio for EDD for problem $1|r_j, d_j < 0|L_{max}$</u>

- structure of a schedule S



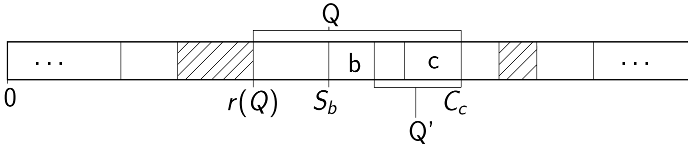
- critical job c of a schedule: job with $L_c = \max L_j$
- critical sequence Q: jobs processed in the interval $[t, C_c]$, where t is the earliest time that the machine is not idle in $[t, C_c]$
- if $q_c = \min_{j \in Q} q_j$ the schedule is optimal since then

$$L_{max}(S) = L_c = C_c - d_c = r(Q) + p(Q) + q(Q) \leq L^*_{max}$$

- Notation: L^*_{max} denotes the optimal value

Approximation ratio for EDD for problem $1|r_j, d_j < 0|L_{max}$

- structure of a schedule



- interference job b: last scheduled job from Q with $q_b < q_c$
- Lemma: For the objective value $L_{max}(EDD)$ of an EDD schedule we have: (Proofs on the board)
  1. $L_{max}(EDD) - L^*_{max} < q_c$
  2. $L_{max}(EDD) - L^*_{max} < p_b$
- Theorem: EDD is 2-approximation algorithm for $1|r_j, d_j < 0|L_{max}$

Single machine models: Maximum Lateness

Approximation ratio for EDD for problem $1|r_j, d_j < 0|L_{max}$

- Remarks:
  - EDD is also a 2-approximation for $1|prec, r_j, d_j < 0|L_{max}$ (uses modified release and due dates)
  - by an iteration technique the approximation factor can be reduced to $3/2$

Single machine models: Maximum Lateness

Enumerative methods for problem $1|r_j|L_{max}$

- we again will use head-body-tail notation
- Simple branch and bound method:
  - branch on level i of the search tree by selecting a job to be scheduled on position i
  - if in a node of the search tree on level i the set of already scheduled jobs is denoted by S and the finishing time of the jobs from S by t, for position i we only have to consider jobs k with

$$r_k < \min_{j \notin S}(\max\{t, r_j\} + p_j)$$

  - lower bound: solve for remaining jobs $1|r_j, pmtn|L_{max}$
  - search strategy: depth first search + selecting next job via lower bound

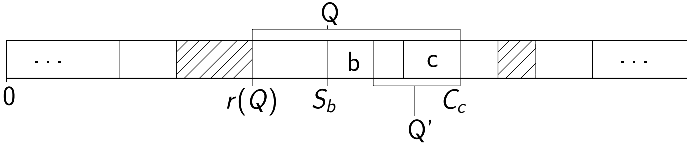
Single machine models: Maximum Lateness

Advanced b&b-methods for problem $1|r_j|L_{max}$

- node of search tree = restricted instance
- restrictions = set of precedence constraints
- branching = adding precedence constraints between certain pairs of jobs
- after adding precedence constraints, modify release and due dates
- apply EDD to instance given in a node
  - critical sequence has no interference job: EDD solves instance optimal
    $\rightarrow$ backtrack
  - critical sequence has an interference job: branch

Advanced b&b-methods for problem $1|r_j|L_{max}$ (cont.)

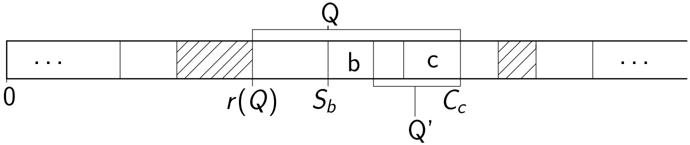
branching, given set Q, critical job c, interference job b, and set Q' of jobs from Q following b



- $L_{max} = S_b + p_b + p(Q') + q(Q') < r(Q') + p_b + p(Q') + q(Q')$
- if b is scheduled between jobs of Q' the value is at least $r(Q') + p_b + p(Q') + q(Q')$; i.e. worse than the current schedule

Advanced b&b-methods for problem $1|r_j|L_{max}$ (cont.)

branching, given sequence Q, critical job c, interference job b, and set Q' of jobs from Q following b



- $L_{max} = S_b + p_b + p(Q') + q(Q') < r(Q') + p_b + p(Q') + q(Q')$
- if b is scheduled between jobs of Q' the value is at least $r(Q') + p_b + p(Q') + q(Q')$; i.e. worse than the current schedule
- branch by adding either $b \rightarrow Q'$ or $Q' \rightarrow b$

Advanced b&b-methods for problem $1|r_j|L_{max}$ (cont.)

- lower bounds in a node: maximum of
  - lower bound of parent node
  - $r(Q') + p(Q') + q(Q')$
  - $r(Q' \cup \{b\}) + p(Q' \cup \{b\}) + q(Q' \cup \{b\})$

  using the modified release and due dates
- upper bound UB: best value of the EDD schedules
- discard a node if lower bound $\geq UB$
- search strategy: select node with minimum lower bound

Advanced b&b-methods for problem $1|r_j|L_{max}$ (cont.)

- speed up possibility:
  - let $k \notin Q' \cup \{b\}$ with $r(Q') + p_k + p(Q') + q(Q') \geq UB$
  - if $r(Q') + p(Q') + p_k + q_k \geq UB$ then add $k \rightarrow Q'$
  - if $r_k + p_k + p(Q') + q(Q') \geq UB$ then add $Q' \rightarrow k$

Single machine models: Number of Tardy Jobs

Problem $1||\sum U_j$:

- Structure of an optimal schedule:
  - set S_1 of jobs meeting their due dates
  - set S_2 of jobs being late
  - jobs of S_1 are scheduled before jobs from S_2
  - jobs from S_1 are scheduled in EDD order
  - jobs from S_2 are scheduled in an arbitrary order
- Result: a partition of the set of jobs into sets S_1 and S_2 is sufficient to describe a solution

Single machine models: Number of Tardy Jobs

Algorithm $1||\sum U_j$

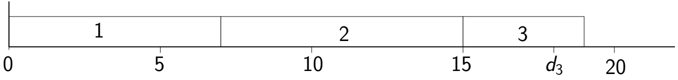
1. enumerate jobs such that $d_1 \leq \ldots \leq d_n$;
2. $S_1 := \emptyset$; $t := 0$;
3. FOR j:=1 TO n DO
4. $\quad S_1 := S_1 \cup \{j\}$; $t := t + p_j$;
5. $\quad$ IF $t > d_j$ THEN
6. $\quad\quad$ Find job k with largest p_k value in S_1;
7. $\quad\quad$ $S_1 := S_1 \setminus \{k\}$; $t := t - p_k$;
8. $\quad$ END
9. END

Remarks Algorithm $1||\sum U_j$

- Principle: schedule jobs in order of increasing due dates and always when a job gets late, remove the job with largest processing time; all removed jobs are late
- complexity: $O(n\log(n))$
- Example: $n = 5$; $p = (7, 8, 4, 6, 6)$; $d = (9, 17, 18, 19, 21)$

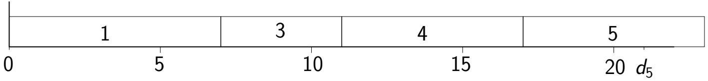
Remarks Algorithm $1||\sum U_j$

- Principle: schedule jobs in order of increasing due dates and always when a job gets late, remove the job with largest processing time; all removed jobs are late
- complexity: $O(n\log(n))$
- Example: $n = 5$; $p = (7, 8, 4, 6, 6)$; $d = (9, 17, 18, 19, 21)$



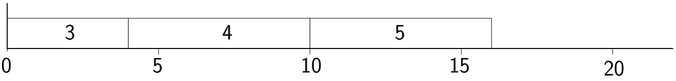
Remarks Algorithm $1||\sum U_j$

- Principle: schedule jobs in order of increasing due dates and always when a job gets late, remove the job with largest processing time; all removed jobs are late
- complexity: $O(n \log(n))$
- Example: $n = 5$; $p = (7, 8, 4, 6, 6)$; $d = (9, 17, 18, 19, 21)$



Remarks Algorithm $1||\sum U_j$

- Principle: schedule jobs in order of increasing due dates and always when a job gets late, remove the job with largest processing time; all removed jobs are late
- complexity: $O(n\log(n))$
- Example: $n = 5$; $p = (7, 8, 4, 6, 6)$; $d = (9, 17, 18, 19, 21)$



- Algorithm $1||\sum U_j$ computes an optimal solution
  Proof on the board

Single machine models: Weighted Number of Tardy Jobs

Problem $1||\sum w_j U_j$

- problem $1||\sum w_j U_j$ is NP-hard even if all due dates are the same; i.e. $1|d_j = d|\sum w_j U_j$ is NP-hard
  Proof on the board (reduction from PARTITION)
- priority based heuristic (WSPT-rule):
  schedule jobs in increasing p_j/w_j order

Single machine models: Weighted Number of Tardy Jobs

Problem $1||\sum w_jU_j$

- problem $1||\sum w_jU_j$ is NP-hard even if all due dates are the same; i.e. $1|d_j = d|\sum w_jU_j$ is NP-hard
  Proof on the board (reduction from PARTITION)
- priority based heuristic (WSPT-rule):
  schedule jobs in increasing p_j/w_j order
- WSPT may perform arbitrary bad for $1||\sum w_jU_j$:

Problem $1||\sum w_j U_j$

- problem $1||\sum w_j U_j$ is NP-hard even if all due dates are the same; i.e. $1|d_j = d|\sum w_j U_j$ is NP-hard
  Proof on the board (reduction from PARTITION)
- priority based heuristic (WSPT-rule):
  schedule jobs in increasing p_j/w_j order
- WSPT may perform arbitrary bad for $1||\sum w_j U_j$:

  $n = 3; p = (1, 1, M);\ \ w = (1 + \epsilon, 1, M - \epsilon); d = (1 + M, 1 + M, 1 + M)$

$$\sum w_j U_j(WSPT)/\sum w_j U_j(opt) = (M - \epsilon)/(1 + \epsilon)$$

Dynamic Programming for $1||\sum w_j U_j$

- assume $d_1 \leq \ldots \leq d_n$
- as for $1||\sum U_j$ a solution is given by a partition of the set of jobs into sets S_1 and S_2 and jobs in S_1 are in EDD order
- Definition:
    - $F_j(t) :=$ minimum criterion value for scheduling the first j jobs such that the processing time of the on-time jobs is at most t
- $F_n(T)$ with $T = \sum_{j=1}^{n} p_j$ is optimal value for problem $1||\sum w_j U_j$
- Initial conditions:

$$F_j(t) = \begin{cases} \infty & \text{for } t < 0;\ j = 1, \ldots, n \\ 0 & \text{for } t \geq 0;\ j = 0 \end{cases} \tag{1}$$

Single machine models: Weighted Number of Tardy Jobs

Dynamic Programming for $1||\sum w_j U_j$ (cont.)

- if $0 \leq t \leq d_j$ and j is late in the schedule corresponding to $F_j(t)$, we have $F_j(t) = F_{j-1}(t) + w_j$
- if $0 \leq t \leq d_j$ and j is on time in the schedule corresponding to $F_j(t)$, we have $F_j(t) = F_{j-1}(t - p_j)$

Single machine models: Weighted Number of Tardy Jobs

Dynamic Programming for $1||\sum w_j U_j$ (cont.)

- if $0 \leq t \leq d_j$ and j is late in the schedule corresponding to $F_j(t)$, we have $F_j(t) = F_{j-1}(t) + w_j$
- if $0 \leq t \leq d_j$ and j is on time in the schedule corresponding to $F_j(t)$, we have $F_j(t) = F_{j-1}(t - p_j)$
- summarizing, we get for $j = 1, \ldots, n$:

$$F_j(t) = \begin{cases} \min\{F_{j-1}(t - p_j), F_{j-1}(t) + w_j\} & \text{for } 0 \leq t \leq d_j \\ F_j(d_j) & \text{for } d_j < t \leq T \end{cases} \tag{2}$$

<u>DP-algorithm for $1||\sum w_j U_j$</u>

1. initialize $F_j(t)$ according to (1)
2. FOR $j := 1$ TO n DO
3. FOR $t := 0$ TO T DO
4. update $F_j(t)$ according to (2)
5. $\sum w_j U_j(OPT) = F_n(d_n)$

Single machine models: Weighted Number of Tardy Jobs

DP-algorithm for $1||\sum w_j U_j$

1. initialize $F_j(t)$ according to (1)
2. FOR $j := 1$ TO n DO
3. FOR $t := 0$ TO T DO
4. update $F_j(t)$ according to (2)
5. $\sum w_j U_j(OPT) = F_n(d_n)$

- complexity is $O(n \sum_{j=1}^{n} p_j)$
- thus, algorithm is pseudopolynomial

Single machine models: Total Tardiness

Basic results:

- $1||\sum T_j$ is NP-hard
- preemption does not improve the criterion value
  $\rightarrow 1|pmtn|\sum T_j$ is NP-hard
- idle times do not improve the criterion value
- Lemma 1: If $p_j \leq p_k$ and $d_j \leq d_k$, then an optimal schedule exist in which job j is scheduled before job k.
  Proof: exercise
- this lemma gives a dominance rule

Single machine models: Total Tardiness

Structural property for $1||\sum T_j$

- let k be a fixed job and $\hat{C}_k$ be latest possible completion time of job k in an optimal schedule
- define

$$\hat{d}_j = \begin{cases} d_j & \text{for } j \neq k \\ \max\{d_k, \hat{C}_k\} & \text{for } j = k \end{cases}$$

- Lemma 2: Any optimal sequence w.r.t. $\hat{d}_1, \ldots, \hat{d}_n$ is also optimal w.r.t. $d_1, \ldots, d_n$.
  Proof on the board

Structural property for $1||\sum T_j$ (cont.)

- let $d_1 \leq \ldots \leq d_n$
- let k be the job with $p_k = \max\{p_1, \ldots, p_n\}$
- Lemma 1 implies that an optimal schedule exists where

$$\{1, \ldots, k-1\} \rightarrow k$$

- Lemma 3: There exists an integer δ, $0 \leq \delta \leq n-k$ for which an optimal schedule exist in which

$$\{1, \ldots, k-1, k+1, \ldots, k+\delta\} \rightarrow k \text{ and } k \rightarrow \{k+\delta+1, \ldots, n\}.$$

Proof on the board

Single machine models: Total Tardiness

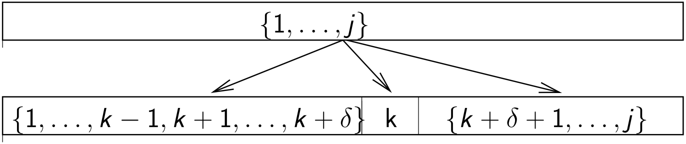
<u>DP-algorithm for $1||\sum T_j$</u>

- Definition:
  - $F_j(t) :=$ minimum criterion value for scheduling the first j jobs starting their processing at time t
- by Lemma 3 we get:
  there exists some $\delta \in \{1, \ldots, j\}$ such that $F_j(t)$ is achieved by scheduling
  1. first jobs $1, \ldots, k-1, k+1, \ldots, k+\delta$ in some order
  2. followed by job k starting at $t + \sum_{l=1}^{k+\delta} p_l - p_k$
  3. followed by jobs $k+\delta+1, \ldots, j$ in some order

  where $p_k = \max_{l=1}^{j} p_l$

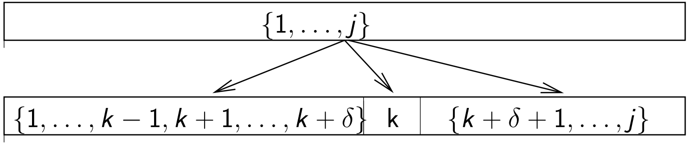
Single machine models: Total Tardiness

DP-algorithm for $1||\sum T_j$ (cont.)

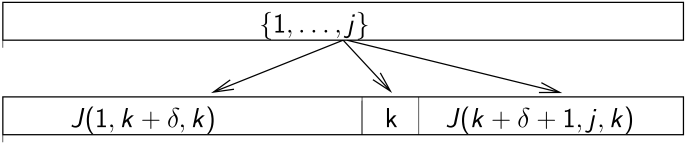


- Definition:
  - $J(j, l, k) := \{i | i \in \{j, j+1, \ldots, l\}; p_i \leq p_k; i \neq k\}$

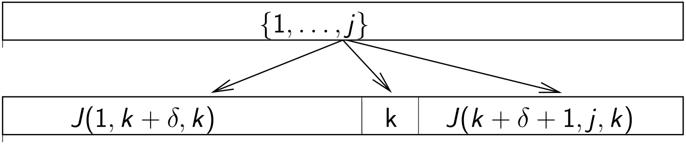
DP-algorithm for $1||\sum T_j$ (cont.)



- Definition:
  - $J(j,l,k) := \{i | i \in \{j, j+1, \ldots, l\}; p_i \leq p_k; i \neq k\}$



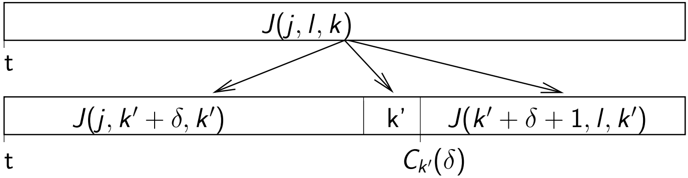
DP-algorithm for $1||\sum T_j$ (cont.)



- Definition:
  - $J(j,l,k) := \{i | i \in \{j, j+1, \ldots, l\}; p_i \leq p_k; i \neq k\}$
  - $V(J(j,l,k),t)$:= minimum criterion value for scheduling the jobs from $J(j,l,k)$ starting their processing at time t

Single machine models: Total Tardiness

DP-algorithm for $1||\sum T_j$ (cont.)



- we get:

$$\begin{aligned} V(J(j,l,k),t) = \min_\delta \quad & \{V(J(j,k'+\delta,k'),t) \\ & + \max\{0, C_{k'}(\delta) - d_{k'}\} \\ & + V(J(k'+\delta+1,l,k'), C_{k'}(\delta)))\} \end{aligned}$$

  where $p_{k'} = \max\{p_{j'} | j' \in J(j,l,k)\}$ and
  $C_{k'}(\delta) = t + p_{k'} + \sum_{j' \in V(J(j,k'+\delta,k'))} p_{j'}$

- $V(\emptyset, t) = 0, \;\; V(\{j\}, t) = \max\{0, t + p_j - d_j\}$

DP-algorithm for $1||\sum T_j$ (cont.)

- optimal value of $1||\sum T_j$ is given by $V(\{1,\dots,n\},0)$
- complexity:
  - at most $O(n^3)$ subsets $J(j,l,k)$
  - at most $\sum p_j$ values for t
  - each recursion (evaluation $V(J(j,l,k),t)$) costs $O(n)$ (at most n values for δ)

  total complexity: $O(n^4 \sum p_j)$ (pseudopolynomial)