DM87
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 8

Single Machine Models

Marco Chiarandini

Outline

1. Dispatching Rules

2. Single Machine Models

Outline

1. Dispatching Rules

2. Single Machine Models

Dispatching rules

Distinguish static and dynamic rules.

- Service in random order (SIRO)
- Earliest release date first (ERD=FIFO)
  - tends to min variations in waiting time
- Earliest due date (EDD)
- Minimal slack first (MS)
  - $j^* = \arg\min_j\{\max(d_j - p_j - t, 0)\}$.
  - tends to min due date objectives (T,L)

- (Weighted) shortest processing time first (WSPT)
  - $j^* = \arg\max_j\{w_j/pj\}$.
  - tends to min $\sum w_j C_j$ and max work in progress and
- Loongest processing time first (LPT)
  - balance work load over parallel machines
- Shortest setup time first (SST)
  - tends to min C_{max} and max throughput
- Least flexible job first (LFJ)
  - eligibility constraints

- Critical path (CP)
  - first job in the CP
  - tends to min C_{max}
- Largest number of successors (LNS)
- Shortest queue at the next operation (SQNO)
  - tends to min idleness of machines

Table C.1. Summary of Dispatching Rules

	RULE	DATA	OBJECTIVES
Rules Dependent on Release Dates and Due Dates	*ERD*	r_j	Variance in Throughput Times
	EDD	d_j	Maximum Lateness
	MS	d_j	Maximum Lateness
Rules Dependent on Processing Times	*LPT*	p_j	Load Balancing over Parallel Machines
	SPT	p_j	Sum of Completion Times, WIP
	WSPT	p_j, w_j	Weighted Sum of Completion Times, WIP
	CP	p_j, *prec*	Makespan
	LNS	p_j, *prec*	Makespan
Miscellaneous	*SIRO*	–	Ease of Implementation
	SST	s_{jk}	Makespan and Throughput
	LFJ	M_j	Makespan and Throughput
	SQNO	–	Machine Idleness

When dispatching rules are optimal?

	RULE	DATA	ENVIRONMENT
1	SIRO	—	—
2	ERD	r_j	$1 \mid r_j \mid \mathrm{Var}(\sum(C_j - r_j)/n)$
3	EDD	d_j	$1 \mid\mid L_{\max}$
4	MS	d_j	$1 \mid\mid L_{\max}$
5	SPT	p_j	$Pm \mid\mid \sum C_j$; $Fm \mid p_{ij} = p_j \mid \sum C_j$
6	WSPT	w_j, p_j	$Pm \mid\mid \sum w_j C_j$
7	LPT	p_j	$Pm \mid\mid C_{\max}$
8	SPT-LPT	p_j	$Fm \mid block, p_{ij} = p_j \mid C_{\max}$
9	CP	$p_j, prec$	$Pm \mid prec \mid C_{\max}$
10	LNS	$p_j, prec$	$Pm \mid prec \mid C_{\max}$
11	SST	s_{jk}	$1 \mid s_{jk} \mid C_{\max}$
12	LFJ	M_j	$Pm \mid M_j \mid C_{\max}$
13	LAPT	p_{ij}	$O2 \mid\mid C_{\max}$
14	SQ	—	$Pm \mid\mid \sum C_j$
15	SQNO	—	$Jm \mid\mid \gamma$

Composite dispatching rules

Why composite rules?

- Example: $1 \,||\, \sum w_j T_j$:
  - WSPT, optimal if due dates are zero
  - EDD, optimal if due dates are loose
  - MS, tends to minimize T

➤ The efficacy of the rules depends on instance factors

Instance characterization

- Job attributes: {weight, processing time, due date, release date}
- Machine attributes: {speed, num. of jobs waiting, num. of jobs eligible }

Possible instance factors:

$$\theta_1 = 1 - \frac{\bar{d}}{c_{max}} \quad \text{(due date tightness)}$$

$$\theta_2 = \frac{d_{max} - d_{min}}{c_{max}} \quad \text{(due date range)}$$

$$\theta_3 = \frac{\bar{s}}{\bar{p}} \quad \text{(set up time severity)}$$

(estimated $\hat{C}_{max} = \sum_{j=1}^{n} p_j + n\bar{s}$)

- Dynamic apparent tardiness cost (ATC)

$$I_j(t) = \frac{w_j}{p_j} \exp\left(-\frac{\max(d_j - p_j - t, 0)}{K\bar{p}}\right)$$

- Dynamic apparent tardiness cost with setups (ATCS)

$$I_j(t, l) = \frac{w_j}{p_j} \exp\left(-\frac{\max(d_j - p_j - t, 0)}{K_1\bar{p}}\right) \exp\left(\frac{-s_{jk}}{K_2\bar{s}}\right)$$

after job l has finished.

Summary

- Scheduling classification
- Solution methods
- Practice with general solution methods
  - Mathematical Programming
  - Constraint Programming
  - Heuristic methods

Remainder on Scheduling

Objectives:

Look closer into scheduling models and learn:

- special algorithms
- application of general methods

Cases:

- Single Machine
- Parallel Machine
- Permutation Flow Shop
- Job Shop
- Resource Constrained Project Scheduling

Outline

1. Dispatching Rules

2. Single Machine Models

Summary

Single Machine Models:

- C_{max} is sequence independent
- if $r_j = 0$ and h_j is monotone in C_j then optimal schedule is nondelay and has no preemption.

$1 \,||\, \sum w_j C_j$

[Total weighted completion time]

- **Theorem:** The weighted shortest processing time first (WSPT) rule is optimal.

Extensions to $1 \,|\, prec \,|\, \sum w_j C_j$

- in the general case strongly NP-hard
- chain precedences:
  process first chain with highest ρ-factor up to, and included, job with highest ρ-factor.
- poly also for tree and sp-graph precedences

Extensions to $1 \mid r_j, prmp \mid \sum w_j C_j$

- in the general case strongly NP-hard
- preemptive version of the WSPT if equal weights
- however, $1 \mid r_j \mid \sum w_j C_j$ is strongly NP-hard

$1 \mid prec \mid L_{max}$

[maximum lateness]

- generalization: $h_{max} = \max\{h(C_1), h(C_2), \ldots, h(C_n)\}$
- Solved by backward dynamic programming in $O(n^2)$:

  J set of jobs already scheduled;
  J^c set of jobs still to schedule;
  $J' \subseteq J^c$ set of schedulable jobs

  Step 1: Set $J = \emptyset$, $J^c = \{1, \ldots, n\}$ and J' the set of all jobs with no successor

  Step 2: Select j^* such that $j^* = \arg\min_{j \in J'}\{h_j\left(\sum_{k \in J^c} p_k\right)\}$;
  add j^* to J; remove j^* from J^c; update J'.

  Step 3: If J^c is empty then stop, otherwise go to Step 2.

- For $1 \mid\mid L_{max}$ Earliest Due Date first
- $1|r_j|L_{max}$ is instead strongly NP-hard

$1 \mid\mid \sum h_j(C_j)$

- generalization of $\sum w_j T_j$ hence strongly NP-hard
- efficient (forward) dynamic programming algorithm $O(2^n)$

  J set of job already scheduled;

  $V(J) = \sum_{j \in J} h_j(C_j)$

  Step 1: Set $J = \emptyset$, $V(j) = h_j(p_j), \quad j = 1, \ldots, n$

  Step 2: $V(J) = \min_{j \in J}\left(V(J - \{j\}) + h_j\left(\sum_{k \in J} p_k\right)\right)$

  Step 3: If $J = \{1, 2, \ldots, n\}$ then $V(\{1, 2, \ldots, n\})$ is optimum, otherwise go to Step 2.

$1 \mid s_{jk} \mid C_{max}$

[Makespan with sequence-dependent setup times]

- general case is NP-hard (traveling salesman reduction).
- special case:

  parameters a_j, b_j for job j with

  $$s_{jk} \propto |a_k - b_j|$$

  [Gilmore and Gomory, 1964] give a $O(n^2)$ algorithm

- assume $b_0 \leq b_1 \leq \ldots \leq b_n$ ($k > j$ and $b_k \geq b_j$)
- one-to-one correspondence with solution of TSP with $n+1$ cities
  city 0 has a_0, b_0
  start at b_0 finish at a_0
- tour representation $\phi : \{0, 1, \ldots, n\} \mapsto \{0, 1, \ldots, n\}$
  (permutation map, single linked array)
- Hence,

$$\min \quad c(\phi) = \sum_{i=1}^{n} c_{i,\phi(i)} \tag{1}$$

$$\phi(S) \neq S \qquad \forall S \subset V \tag{2}$$

- find ϕ^* by ignoring (2)
  make ϕ^* a tour through swaps
  (swap chosen solving a min spanning tree and applied in a certain order)

- Interchange δ^{jk}

$$\delta^{jk}(\phi) = \{\phi' \quad | \quad \phi'(j) = \phi(k), \quad \phi(k) = \phi(j), \quad \phi'(l) = \phi(l), \quad \forall l \neq j, k\}$$

- Cost

$$\begin{aligned} c_\phi(\delta^{jk}) &= c(\delta^{jk}(\phi)) - c(\phi) \\ &= \| [b_j, b_k] \cap [a_{\phi(j)}, a_{\phi(k)}] \| \end{aligned}$$

- **Theorem:** Let ϕ^* be a permutation that ranks the a that is $k > j$ implies $a_{\phi(k)} \geq a_{\phi(j)}$ then

$$c(\phi^*) = \min_\phi c(\phi)$$

.

- **Lemma:** If ϕ is a permutation consisting of cycles $C_1, \ldots, C_p$ and δ^{jk} is an interchange with $j \in C_r$ and $k \in C_s$, $r \neq s$, then $\delta^{jk}(\phi)$ contains the same cycles except that C_r and C_s have been replaced by a single cycle containing all their nodes.

- **Theorem:** Let $\delta^{j_1 k_1}, \delta^{j_2 k_2}, \ldots, \delta^{j_p k_p}$ be the interchanges corresponding to the arcs of a spanning tree of G_{ϕ^*}. The arcs may be taken in any order. Then ϕ',

$$\phi' = \delta^{j_1 k_1} \circ \delta^{j_2 k_2} \circ \ldots \circ \delta^{j_p k_p}(\phi^*)$$

  is a tour.
- The $p-1$ interchanges can be found by greedy algorithm
  (similarity to Kruskal for min spanning tree)
- **Lemma:** There is a minimum spanning tree in G_{ϕ^*} that contains only arcs $\delta^{j,j+1}$.
- Generally, $c(\phi') \neq c(\delta^{j_1 k_1}) + c(\delta^{j_2 k_2}) + \ldots + c(\delta^{j_p k_p})$.

-

$$\text{node } j \text{ in } \phi \text{ is of } \begin{cases} \text{Type I,} & \text{if } b_j \leq a_{\phi(j)} \\ \text{Type II,} & \text{otherwise} \end{cases}$$

$$\text{interchange } jk \text{ is of } \begin{cases} \text{Type I,} & \text{if lower node of type I} \\ \text{Type II,} & \text{if lower node of type II} \end{cases}$$

- Order:
  interchanges in Type I in decreasing order
  interchanges in Type II in increasing order
- Apply to ϕ^* interchanges of Type I and Type II in that order.
- **Theorem:** The tour found is a minimal cost tour.

Resuming the final algorithm [Gilmore and Gomory, 1964]:

Step 1: Arrange b_j in order of size and renumber jobs so that $b_j \leq b_{j+1}$, $j = 1, \ldots, n$.

Step 2: Arrange a_j in order of size.

Step 3: Define ϕ by $\phi(j) = k$ where k is the $j+1$-smallest of the a_j.

Step 4: Compute the interchange costs $c_{\delta^{j,j+1}}$, $j = 0, \ldots, n-1$

$$c_{\delta^{j,j+1}} = \| [b_j, b_{j+1}] \cap [a_{\phi(j)}, a_{\phi(i)}] \|$$

Step 5: While G has not one single component, Add to G_ϕ the arc of minimum cost $c(\delta^{j,j+1})$ such that j and $j+1$ are in two different components.

Step 6: Divide the arcs selected in Step 5 in Type I and II.
Sort Type I in decreasing and Type II increasing order of index.
Apply the relative interchanges in the order.

Summary

Single Machine Models:

$1 \,|\,|\, \sum w_j C_j$: weighted shortest processing time first is optimal

$1 \,|\, prec \,|\, L_{max}$: dynamic programming in $O(n^2)$

$1 \,|\,|\, \sum h_j(C_j)$: dynamic programming in $O(2^n)$

$1 \,|\, s_{jk} \,|\, C_{max}$: in the special case, Gilmore and Gomory algorithm optimal in $O(n^2)$