DM204, 2010
SCHEDULING, TIMETABLING AND ROUTING

Lecture 12
Single Machine Models, Dynamic Programming

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

1. Dispatching Rules

2. Single Machine Models

Organization

- Next week exercise session on Monday or Wednesday?
- Tomorrow: midterm evaluation
- Project to be launched beginning next week
- All elements for tackling it have been given during the past lectures
- Lectures also during the break?
- How the course will continue
  We will look closer into scheduling models and learn:
  - special algorithms
  - application of general methods

Course Overview

- ✔ Problem Introduction
  - ✔ Scheduling classification
  - ✔ Scheduling complexity
  - ✔ RCPSP
- General Methods
  - ✔ Integer Programming
  - ✔ Constraint Programming
  - ✔ Heuristics
  - Dynamic Programming
  - Branch and Bound
- Scheduling
  - Single Machine
  - Parallel Machine and Flow Shop Models
  - Job Shop
  - Resource Constrained Project Scheduling Model
- Timetabling
  - Reservations and Education
  - University Timetabling
  - Crew Scheduling
  - Public Transports
- Vechicle Routing
  - Capacited Models
  - Time Windows models
  - Rich 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

Dispatching Rules in Scheduling

	RULE	DATA	OBJECTIVES
Rules Dependent	ERD	r_j	Variance in Throughput Times
on Release Dates	EDD	d_j	Maximum Lateness
and Due Dates	MS	d_j	Maximum Lateness
	LPT	p_j	Load Balancing over Parallel Machines
Rules Dependent	SPT	p_j	Sum of Completion Times, WIP
on Processing	WSPT	p_j, w_j	Weighted Sum of Completion Times, WIP
Times	CP	p_j, $prec$	Makespan
	LNS	p_j, $prec$	Makespan
	SIRO	-	Ease of Implementation
Miscellaneous	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: $1 \mid\mid \sum w_j T_j$:
  - 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:
  - $1 \mid\mid \sum w_j T_j$

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

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

  - $1 \mid s_{jk} \mid \sum w_j T_j$

$$(\theta_1, \theta_2 \text{ with estimated } \hat{C}_{max} = \sum_{j=1}^{n} p_j + n\bar{s})$$

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

- $1 \,|\,|\, \sum w_j T_j$, dynamic apparent tardiness cost (ATC)

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

- $1 \,|\, s_{jk} |\, \sum w_j T_j$, 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.

Outline

1. Dispatching Rules

2. Single Machine Models

Outlook

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

$1 \,|\,|\, \sum_j U_j$: Moore's algorithm

$1 \,|\, prec |\, L_{max}$: Lawler's algorithm, backward dynamic programming in $O(n^2)$ [Lawler, 1973]

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

$1 \,|\,|\, \sum w_j T_j$: local search and dynasearch

$1 \,|\, r_j, (prec) \,|\, L_{max}$: branch and bound

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

$1 \,|\,|\, \sum w_j T_j$: column generation approaches

Multicriteria

Summary

Single Machine Models:

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

$1 \mid \mid \sum w_j C_j$

[Total weighted completion time]

Theorem

The weighted shortest processing time first (WSPT) rule is optimal.

Extensions to $1 \mid prec \mid \sum w_j C_j$

- 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.
- polytime algorithm 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

$1 \mid \mid \sum_j U_j$

[Number of tardy jobs]

- [Moore, 1968] algorithm in $O(n \log n)$
  - Add jobs in increasing order of due dates
  - If inclusion of job j^* results in this job being completed late discard the scheduled job k^* with the longest processing time
- $1 \mid \mid \sum_j w_j U_j$ is a knapsack problem hence NP-hard

Dynamic programming

Procedure based on divide and conquer

Principle of optimality the completion of an optimal sequence of decisions must be optimal

- Break down the problem into stages at which the decisions take place
- Find a recurrence relation that takes us backward (forward) from one stage to the previous (next)
- Typical technique: labelling with dominance criteria

(In scheduling, backward procedure feasible only if the makespan is schedule independent, eg, single machine problems without setups, multiple machines problems with identical processing times.)

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

- $h_{max} = \max\{h_1(C_1), h_2(C_2), \ldots, h_n(C_n)\}$, h_j regular
- special case: $1 \mid prec \mid h_{max}$ [maximum lateness]
- solved by backward dynamic programming in $O(n^2)$ [Lawler, 1978]

  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
- (forward) dynamic programming algorithm $O(2^n)$

  J set of jobs 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.

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

A lot of work done on $1 \mid \mid \sum w_j T_j$
[single-machine total weighted tardiness]

- $1 \mid \mid \sum T_j$ is hard in ordinary sense, hence admits a pseudo polynomial algorithm (dynamic programming in $O(n^4 \sum p_j)$)
- $1 \mid \mid \sum w_j T_j$ strongly NP-hard (reduction from 3-partition)
  - exact solution via branch and bound feasible up to 40 jobs [Potts and Wassenhove, Oper. Res., 1985]
  - exact solution via time-indexed integer programming formulation used to lower bound in branch and bound solves instances of 100 jobs in 4-9 hours [Pan and Shi, Math. Progm., 2007]
  - dynasearch: results reported for 100 jobs within a 0.005% gap from optimum in less than 3 seconds [Grosso et al., Oper. Res. Lett., 2004]

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

Local search (revisited)

1. search space (solution representation)
2. initial solution
3. neghborhood function
4. evaluation function
5. step function
6. memory states
7. termination predicte

Speedups Techniques for Efficient Neighborhood Search

1) Incremental updates

2) Neighborhood pruning

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

Neighborhood updates and pruning

- Interchange neigh.: size $\binom{n}{2}$ and $O(|i-j|)$ evaluation each
  - first-improvement: π_j, π_k
    - $p_{\pi_j} \leq p_{\pi_k}$ for improvements, $w_j T_j + w_k T_k$ must decrease because jobs in $\pi_j, \ldots, \pi_k$ can only increase their tardiness.
    - $p_{\pi_j} \geq p_{\pi_k}$ possible use of auxiliary data structure to speed up the computation
  - best-improvement: π_j, π_k
    - $p_{\pi_j} \leq p_{\pi_k}$ for improvements, $w_j T_j + w_k T_k$ must decrease at least as the best interchange found so far because jobs in $\pi_j, \ldots, \pi_k$ can only increase their tardiness.
    - $p_{\pi_j} \geq p_{\pi_k}$ possible use of auxiliary data structure to speed up the computation
- Swap: size $n-1$ and $O(1)$ evaluation each
- Insert: size $(n-1)^2$ and $O(|i-j|)$ evaluation each
  But possible to speed up with systematic examination by means of swaps: an interchange is equivalent to $|i-j|$ swaps hence overall examination takes $O(n^2)$

Dynasearch

- two interchanges δ_{jk} and δ_{lm} are independent if $\max\{j,k\} < \min\{l,m\}$ or $\min\{l,k\} > \max\{l,m\}$;
- the dynasearch neighborhood is obtained by a series of independent interchanges;
- it has size $2^{n-1} - 1$;
- but a best move can be found in $O(n^3)$ searched by dynamic programming;
- it yields in average better results than the interchange neighborhood alone.

Table 1 Data for the Problem Instance

Job j	1	2	3	4	5	6
Processing time p_j	3	1	1	5	1	5
Weight w_j	3	5	1	1	4	4
Due date d_j	1	5	3	1	3	1

Table 2 Swaps Made by Best-Improve Descent

Iteration	Current Sequence	Total Weighted Tardiness
	1 2 3 4 5 6	109
1	1 2 3 5 4 6	90
2	1 2 3 5 6 4	75
3	5 2 3 1 6 4	70

Table 3 Dynasearch Swaps

Iteration	Current Sequence	Total Weighted Tardiness
	1 2 3 4 5 6	109
1	1 3 2 5 4 6	89
2	1 5 2 3 6 4	68
3	5 1 2 3 6 4	67

- state (k, π)
- π_k is the partial sequence at state (k, π) that has min $\sum wT$
- π_k is obtained from state (i, π)

$$\begin{cases} \text{appending job } \pi(k) \text{ after } \pi(i) & i = k-1 \\ \text{appending job } \pi(k) \text{ and interchanging } \pi(i+1) \text{ and } \pi(k) & 0 \leq i < k-1 \end{cases}$$

- $F(\pi_0) = 0; \qquad F(\pi_1) = w_{\pi(1)} \left(p_{\pi(1)} - d_{\pi(1)}\right)^+;$

$$F(\pi_k) = \min \begin{cases} F(\pi_{k-1}) + w_{\pi(k)} \left(C_{\pi(k)} - d_{\pi(k)}\right)^+, \\ \min\limits_{1 \leq i < k-1} \{F(\pi_i) + w_{\pi(k)} \left(C_{\pi(i)} + p_{\pi(k)} - d_{\pi(k)}\right)^+ + \\ \quad + \sum_{j=i+2}^{k-1} w_{\pi(j)} \left(C_{\pi(j)} + p_{\pi(k)} - p_{\pi(i+1)} - d_{\pi(j)}\right)^+ + \\ \quad + w_{\pi(i+1)} \left(C_{\pi(k)} - d_{\pi(i+1)}\right)^+ \} \end{cases}$$

- The best choice is computed by recursion in $O(n^3)$ and the optimal series of interchanges for $F(\pi_n)$ is found by backtrack.

- Local search with dynasearch neighborhood starts from an initial sequence, generated by ATC, and at each iteration applies the best dynasearch move, until no improvement is possible (that is, $F(\pi_n^t) = F(\pi_n^{(t-1)})$, for iteration t).

- Speedups:
  - pruning with considerations on $p_{\pi(k)}$ and $p_{\pi(i+1)}$
  - maintainig a string of late, no late jobs
  - h_t largest index s.t. $\pi^{(t-1)}(k) = \pi^{(t-2)}(k)$ for $k = 1, \ldots, h_t$ then $F(\pi_k^{(t-1)}) = F(\pi_k^{(t-2)})$ for $k = 1, \ldots, h_t$ and at iter t no need to consider $i < h_t$.

Dynasearch, refinements:

- [Grosso et al. 2004] add insertion moves to interchanges.

- [Ergun and Orlin 2006] show that dynasearch neighborhood can be searched in $O(n^2)$.

Performance:

- exact solution via branch and bound feasible up to 40 jobs [Potts and Wassenhove, Oper. Res., 1985]

- exact solution via time-indexed integer programming formulation used to lower bound in branch and bound solves instances of 100 jobs in 4-9 hours [Pan and Shi, Math. Progm., 2007]

- dynasearch: results reported for 100 jobs within a 0.005% gap from optimum in less than 3 seconds [Grosso et al., Oper. Res. Lett., 2004]

Summary

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

$1 \mid\mid \sum_j U_j$: Moore's algorithm

$1 \mid prec \mid L_{max}$: Lawler's algorithm, backward dynamic programming in $O(n^2)$ [Lawler, 1973]

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

$1 \mid\mid \sum w_j T_j$: local search and dynasearch