DM87
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 9

Single and Parallel Machine Models

Marco Chiarandini

Outline

1. Single Machine Models

2. Parallel Machine Models

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

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

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

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

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

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

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

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

$1 \,||\, \sum w_j T_j$ branch and Bound, Dynasearch

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

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

$Pm \,|\, prmp \,|\, C_{max}$ Linear Programming, dispatching rules

Outline

1. Single Machine Models

2. Parallel Machine Models

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

[Makespan with sequence-dependent setup]
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.

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

[Maximum lateness with release dates]

- Strongly NP-hard (reduction from 3-partition)
- might have optimal schedule which is not non-delay
- Branch and bound algorithm (valid also for $1 \mid r_j, prec \mid L_{max}$)
  - **Branching**:
    schedule from the beginning (level k, $n!/(k-1)!$ nodes)
    elimination criterion: do not consider job j_k if:

    $$r_j > \min_{l \in J} \{\max(t, r_l) + p_l\} \qquad J \text{ jobs to schedule, } t \text{ current time}$$

  - **Lower bounding**: relaxation to preemptive case for which EDD is optimal

Branch and Bound

S root of the branching tree

LIST := {S};
U:=value of some heuristic solution;
current_best := heuristic solution;
while LIST $\neq \emptyset$
  Choose a branching node k from LIST;
  Remove k from LIST;
  Generate children child(i), $i = 1, \ldots, n_k$, and calculate corresponding lower bounds LB_i;
  for i:=1 to n_k
    if $LB_i < U$ then
      if child(i) consists of a single solution then
        U:=LB_i;
        current_best:=solution corresponding to child(i)
      else add child(i) to LIST

$1||\sum_j U_j$

[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
- $1||\sum_j w_j U_j$ is a knapsack problem hence NP-hard

$1||\sum w_j T_j$

[single-machine total weighted tardiness]

- $1||\sum T_j$ is hard in ordinary sense, hence admits a pseudo polynomial algorithm (dynamic programming)
- $1||\sum w_j T_j$ strongly NP-hard
  - branch and bound
  - time indexed integer program
  - dynaserach

Branch and bound

- **Branching:**
  - work backward in time
  - elimination criterion: if $p_j \leq p_k$ and $d_j \leq d_k$ and $w_j \geq w_k$ then there is an optimal schedule with j before k
- **Lower Bounding:**
  relaxation to preemptive case
  transportation problem

$$\min \sum_{j=1}^{n} \sum_{t=1}^{C_{max}} c_{jt} x_{jt}$$

$$\text{s.t.} \sum_{t=1}^{C_{max}} x_{jt} = p_j, \qquad \forall j = 1, \dots, n$$

$$\sum_{j=1}^{n} x_{jt} \leq 1, \qquad \forall t = 1, \dots, C_{max}$$

$$x_{jt} \geq 0 \qquad \forall j = 1, \dots, n; \; t = 1, \dots, C_{max}$$

[Pan and Shi, 2007]'s lower bounding through time indexed
Stronger but computationally more expensive

$$\min \sum_{j=1}^{n} \sum_{t=1}^{T-p_j} h_j(t+p_j) y_{jt}$$

$$\text{s.t.} \sum_{t=1}^{T-p_j} y_{jt} = 1, \qquad \forall j = 1, \dots, n$$

$$\sum_{j=1}^{n} \sum_{s=t-p_j+1}^{t} y_{jt} \leq 1, \qquad \forall t = 1, \dots, C_{max}$$

$$y_{jt} \geq 0 \qquad \forall j = 1, \dots, n; \; t = 1, \dots, C_{max}$$

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 $O(n^3)$.
- It yields in average better results than the interchange neighborhood alone.
- Searched by dynamic programming

- 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)

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

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

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

- 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]

Complexity resume

Single machine, single criterion problems $1 \mid \mid \gamma$:

C_{max}	$\mathcal{P}$
T_{max}	$\mathcal{P}$
L_{max}	$\mathcal{P}$
h_{max}	$\mathcal{P}$
$\sum C_j$	$\mathcal{P}$
$\sum w_j C_j$	$\mathcal{P}$
$\sum U$	$\mathcal{P}$
$\sum w_j U_j$	weakly $\mathcal{NP}$-hard
$\sum T$	weakly $\mathcal{NP}$-hard
$\sum w_j T_j$	strongly $\mathcal{NP}$-hard
$\sum h_j(C_j)$	strongly $\mathcal{NP}$-hard

Extensions

Non regular objectives

- $1 \mid d_j = d \mid \sum E_j + \sum T_j$
- In an optimal schedule,
  - early jobs are scheduled according to LPT
  - late jobs are scheduled according to SPT

Multicriteria scheduling

Resolution process and decision maker intervention:

- a priori methods (definition of weights, importance)
  - goal programming
  - weighted sum
  - ...
- interactive methods
- a posteriori methods (Pareto optima)
  - lexicographic with goals
  - ...

Outline

1. Single Machine Models

2. Parallel Machine Models

$Pm \,||\, C_{max}$ (without Preemption)

$Pm \,||\, C_{max}$ LPT heuristic, approximation ratio: $\frac{4}{3} - \frac{1}{3m}$

$P\infty \,||\, C_{max}$ CPM

$Pm \,|\, prec \,|\, C_{max}$ strongly NP-hard, LNS heuristic (non optimal)

$Pm \,|\, p_j = 1, M_j \,|\, C_{max}$ LFJ-LFM heuristic (if M_j are nested, then LFJ is optimal)

$Pm \,|\, prmp \,|\, C_{max}$

Not NP hard:

- Linear Programming, x_{ij}: time job j in machine i
- Construction based on lower bound

$$LWB = \max\left\{p_1, \sum_{j=1}^{n} \frac{p_j}{m}\right\}$$

- Dispatching rule: longest remaining processing time (LRPT) optimal in discrete time

$Qm \,|\, prmp \,|\, C_{max}$

- Construction based on

$$LWB = \max\left\{\frac{p_1}{v_1}, \frac{p_1 + p_2}{v_1 + v_2}, \ldots, \frac{\sum_{j=1}^{n} p_j}{\sum_{j=1}^{m} v_j}\right\}$$

- Dispatching rule: longest remaining processing time on the fastest machine first (processor sharing) optimal in discrete time