DM87
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 10

Parallel Machine and Flow Shop Models

Marco Chiarandini

Outline

Outline

Complexity resume

Single machine models

$1 \mid\mid C_{max}$	$\mathcal{P}$	
$1 \mid s_{jk} \mid C_{max}$	$\mathcal{P}$	Gilmore and Gomory's alg. in $O(n^2)$
$1 \mid\mid T_{max}$	$\mathcal{P}$	
$1 \mid\mid L_{max}$	$\mathcal{P}$	
$1 \mid prec \mid L_{max}$	$\mathcal{P}$	Lawler's alg. (Backward dyn. progr.)
$1 \mid r_j, (prec) \mid L_{max}$	strongly $\mathcal{NP}$-hard	Branch and Bound
$1 \mid\mid h_{max}$	$\mathcal{P}$	
$1 \mid\mid \sum C_j$	$\mathcal{P}$	
$1 \mid\mid \sum w_j C_j$	$\mathcal{P}$	WSPT
$1 \mid\mid \sum U$	$\mathcal{P}$	Moore's algorithm
$1 \mid\mid \sum w_j U_j$	weakly $\mathcal{NP}$-hard	
$1 \mid\mid \sum T$	weakly $\mathcal{NP}$-hard	
$1 \mid\mid \sum w_j T_j$	strongly $\mathcal{NP}$-hard	Branch and Bound, Dynasearch
$1 \mid\mid \sum h_j(C_j)$	strongly $\mathcal{NP}$-hard	Dynamic programming in $O(2^n)$

Branch and Bound

[Jens Clausen (1999). Branch and Bound Algorithms - Principles and Examples.]

- Eager Strategy:
  based on the bound value of the subproblems
  1. select a node
  2. branch
  3. for each subproblem compute bounds and compare with current best solution
  4. discard or store nodes together with their bounds

  (Bounds are calculated as soon as nodes are available)
- Lazy Strategy:
  often used when selection criterion for next node is max depth
  1. select a node
  2. compute bound
  3. branch
  4. store the new nodes together with the bound of the processed node

Components

- Initial good feasible solution (heuristic) – might be crucial!
- Bounding function
- Strategy for selecting
- Branching

Bounding

$$\min_{s \in P} g(s) \leq \left\{ \begin{array}{l} \min_{s \in P} f(s) \\ \min_{s \in S} g(s) \end{array} \right\} \leq \min_{s \in S} f(s)$$

P: candidate solutions; $S \subseteq P$ feasible solutions

- relaxation: $\min_{s \in P} f(s)$
- solve (to optimality) in P but with g

Strategy for selecting next subproblem

- best first
  (combined with eager strategy)
- breadth first
  (memory problems)
- depth first
  works on recursive updates (hence good for memory)
  but might compute a large part of the tree which is far from optimal
  (enhanced by alternating search in lowest and largest bounds combined with branching on the node with the largest difference in bound between the children)
  (it seems to perform best)

Branch and bound *vs* backtracking

- $=$ a state space tree is used to solve a problem.
- $\neq$ branch and bound does not limit us to any particular way of traversing the tree (backtracking is depth-first)
- $\neq$ branch and bound is used only for optimization problems.

Branch and bound *vs* A*

- $=$ In A* the admissible heuristic mimics bounding
- $\neq$ In A* there is no branching. It is a search algorithm.
- $\neq$ A* is best first

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)

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

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

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

- 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 $p_{\pi(i+1)}$
  - maintaining 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]

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
  - tardy 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. Resume and Extensions on Single Machine Models

2. Parallel Machine Models

3. Flow Shop

$Pm \mid\mid C_{max}$ (without Preemption)

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

$P\infty \mid prec \mid C_{max}$ CPM

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

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

$Pm \mid prmp \mid C_{max}$

Not NP hard:

- Linear Programming, x_{ij}: time job j in machine i
- Construction based on $LWB = \max\left\{p_1, \sum_{j=1}^{n} \frac{p_j}{m}\right\}$
- Dispatching rule: longest remaining processing time (LRPT) optimal in discrete time

$Qm \mid prmp \mid C_{max}$

- 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

Outline

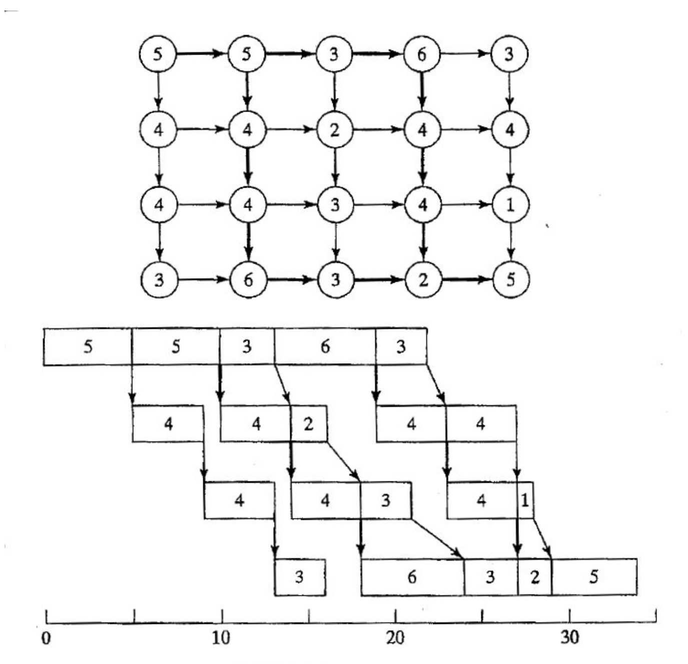
1. Resume and Extensions on Single Machine Models

2. Parallel Machine Models

3. Flow Shop

Flow Shop

- Buffer limited, unlimited
- Permutation Flow Shop
- Directed graph representation
- C_{max} computation (critical path length)



Exact Solutions

- **Theorem:** There always exist an optimum without sequence change in the first two and last two machines.
  (hence $F2 | | C_{max}$ and $F3 | | C_{max}$ are permutation flow shop)
- $F2 | | C_{max}$: Johnson's rule (1954)
  - Set I: $p_{1j} < p_{2j}$, order in increasing p_{1j}, SPT(1)
  - Set II: $p_{2j} < p_{1j}$, order in decreasing p_{2j}, LPT(2)
- $F3 | | C_{max}$ is strongly NP-hard

$Fm \mid prmu, p_{ij} = p_j \mid C_{max}$

[Proportionate permutation flow shop]

- **Theorem:** $C_{max} = \sum_{j=1}^{n} p_j + (m-1)\max(p_1, \dots, p_n)$ and is sequence independent
- Generalization to include machines with different speed: $p_{ij} = p_j / v_i$

  Theorem:
  if the first machine is the bottleneck then LPT is optimal.
  if the last machine is the bottleneck then SPT is optimal.

Construction Heuristics for $Fm \mid prmu \mid C_{max}$

Slope heuristic

- schedule in decreasing order of $A_j = -\sum_{i=1}^{m}(m - (2i-1))p_{ij}$

Campbell, Dudek and Smith's heuristic (1970)

extension of Johnson's rule to when permutation is not dominant

- recursively create 2 machines 1 and $m-1$

$$p'_{ij} = \sum_{k=1}^{i} p_{kj} \qquad p''_{ij} = \sum_{k=m-i+1}^{m} p_{kj}$$

  and use Johnson's rule
- repeat for all $m-1$ possible pairings
- return the best for the overall m machine problem

Nawasz, Enscore, Ham's heuristic (1983)

- **Step 1**: order in decreasing $\sum_{j=1}^{m} p_{ij}$
- **Step 2**: schedule the first 2 jobs at best
- **Step 3**: insert all others in best position

Implementation in $O(n^2 m)$

Framinan, Gupta, Leisten (2004) examined 177 different arrangements of jobs in Step 1 and concluded that the NEH arrangement is the best one for C_{max}.

Metaheuristics for $Fm \mid prmu \mid C_{max}$

Iterated Greedy [Ruiz, Stützle, 2007]

- Destruction: remove d jobs at random
- Construction: reinsert them with NEH heuristic in the order of removal
- Local Search: insertion neighborhood
  (first improvement, whole evaluation $O(n^2 m)$)
- Acceptance Criterion: random walk, best, SA-like

Performance on up to $n = 500 \times m = 20$:

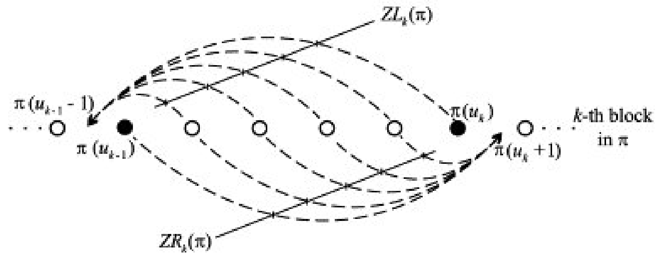
- NEH average gap 3.35% in less than 1 sec.
- IG average gap 0.44% in about 360 sec.

Tabu Search

[Novicki, Smutnicki, 1994, Grabowski, Wodecki, 2004]

- C_{max} expression through critical path
- Block B_k, definition
- Internal block B_k^{Int}, definition
- **Theorem:** Let $\pi, \pi' \in \Pi$, if π' has been obtained from π by an interchange of jobs so that $C_{max}(\pi') < C_{max}(\pi)$ then in π':
  - a) at least one job $j \in B_k$ precedes job $\pi(u_{k-1}), k = 1, \ldots, m$
  - b) at least one job $j \in B_k$ succeeds job $\pi(u_k), k = 1, \ldots, m$

- Insert neighborhood
- Tabu search requires a best strategy. How to search efficiently?
- **Theorem:** (Elimination Criterion) If π' is obtained by π by a "block insertion" then $C_{max}(\pi') \leq C_{max}(\pi)$.
- Define good moves:



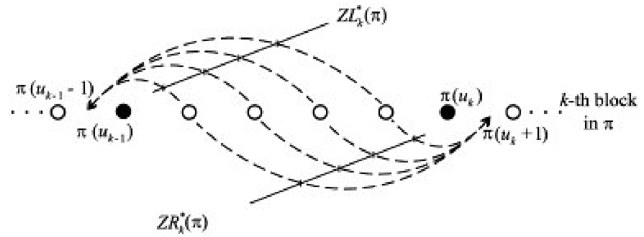
- Use of lower bounds in delta evaluations:

$$D_{ka}(x) = \begin{cases} p_{\pi(x),k+1} - p_{\pi(u_k),k+1} & x \neq u_{k-1} \\ p_{\pi(x),k+1} - p_{\pi(u_k),k+1} + p_{\pi(u_{k-1}+1,k} - p_{\pi(x),k} & x = u_{k-1} \end{cases}$$

$$C_{max}(\delta_x(\pi)) \geq C_{max}(\pi) + D_{ka}(x)$$

- Prohibition criterion:
  an insertion δ_{x,u_k} is tabu if it restores the realtive order of $\pi(x)$ and $\pi(x+1)$.
- Tabu length: $TL = 6 + \left[\frac{n}{10m}\right]$

- Perturbation



  - perform all interchanges among all the blocks that have $D < 0$
  - activated after `MaxIdleIter` idle iterations

Tabu Search: the final algorithm:

Initialization	: $\pi = \pi_0$, $C^* = C_{max}(\pi)$, set iteration counter to zero.
Searching	: Create UR_k and UL_k (set of non tabu moves)
Selection	: Find the best move according to lower bound D. Compute $C_{max}(\delta(\pi))$. Apply move. If improving compare with C^* and in case update. Else increase number of idle iterations.
Stop criterion	: Exit if `MaxIter` iterations are done.
Perturbation	: Apply perturbation if `MaxIdleIter` done.