Outline

1. Parallel Machine Models

2. Flow Shop
   - Introduction
   - Makespan calculation
   - Johnson’s algorithm
   - Construction heuristics
   - Iterated Greedy
   - Efficient Local Search and Tabu Search

Identical machines

Min makespan, without preemption

\[ \text{Pm} | \text{p}_j = 1, \text{M}_j | \text{C}_{\text{max}}: \] least flexible job (LFJ) - least flexible machine (LFM) (optimal if \( \text{M}_j \) are nested)

\[ \text{Pm} | \text{p}_j | \text{C}_{\text{max}}: \] LPT heuristic, approximation ratio: \( \frac{4}{3} - \frac{1}{3m} \)

\[ \text{P\infty} | \text{prec} | \text{C}_{\text{max}}: \] CPM

\[ \text{Pm} | \text{prec} | \text{C}_{\text{max}}: \] strongly NP-hard, LNS heuristic (non optimal)
**Identical machines**

Min makespan, with preemption

\[ Pm \mid \| C_{\text{max}} \]  Not NP-hard:

- Linear Programming (exercise)

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

- Construction based on \( LWB \)

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

**Uniform machines**

Min makespan, with preemption

\[ Qm \mid \text{prmp} \mid C_{\text{max}} \]  Not NP-hard:

- Construction based on

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

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

\( R \parallel \sum_j C_j \) is NP-hard
Solved by local search methods.

- Solution representation
  - a collection of \( m \) sequences, one for each job

Recall that \( 1 \parallel \sum w_j C_j \) is solvable in \( O(n \log n) \)
Unrelated machines

\( R || \sum_j C_j \) is NP-hard
Solved by local search methods.

- Solution representation
  - a collection of \( m \) sequences, one for each job
  - recall that \( 1 \mid \sum w_j C_j \) is solvable in \( O(n \log n) \)
  - indirect representation
    assignment of jobs to machines
    the sequencing is left to the optimal SWPT rule
- Neighborhood: one exchange, swap

Evaluation function. How costly is the computation?

Outline

1. Parallel Machine Models
2. Flow Shop
   - Introduction
   - Makespan calculation
   - Johnson’s algorithm
   - Construction heuristics
   - Iterated Greedy
   - Efficient Local Search and Tabu Search

1. Parallel Machine Models
2. Flow Shop
   - Introduction
   - Makespan calculation
   - Johnson’s algorithm
   - Construction heuristics
   - Iterated Greedy
   - Efficient Local Search and Tabu Search
Flow Shop

General Shop Scheduling:
- $J = \{1, \ldots, N\}$ set of jobs; $M = \{1, 2, \ldots, m\}$ set of machines
- $J_j = \{O_{ij} \mid i = 1, \ldots, n_j\}$ set of operations for each job
- $p_{ij}$ processing times of operations $O_{ij}$

Flow Shop

General Shop Scheduling:
- $J = \{1, \ldots, N\}$ set of jobs; $M = \{1, 2, \ldots, m\}$ set of machines
- $J_j = \{O_{ij} \mid i = 1, \ldots, n_j\}$ set of operations for each job
- $p_{ij}$ processing times of operations $O_{ij}$
- $\mu_{ij} \subseteq M$ machine eligibilities for each operation

Parallel Machine Models

Flow Shop

Introduction

Makespan Problems

Johnson's algorithm

Construction heuristics

Iterated Greedy

Efficient LS and TS
Flow Shop

General Shop Scheduling:
- \( J = \{1, \ldots, N\} \) set of jobs; \( M = \{1, 2, \ldots, m\} \) set of machines
- \( J_j = \{O_{ij} \mid i = 1, \ldots, n_j\} \) set of operations for each job
- \( p_{ij} \) processing times of operations \( O_{ij} \)
- \( \mu_{ij} \subseteq M \) machine eligibilities for each operation
- precedence constraints among the operations
- one job processed per machine at a time,
  one machine processing each job at a time
- \( C_j \) completion time of job \( j \)

Flow Shop

General Shop Scheduling:
- \( J = \{1, \ldots, N\} \) set of jobs; \( M = \{1, 2, \ldots, m\} \) set of machines
- \( J_j = \{O_{ij} \mid i = 1, \ldots, n_j\} \) set of operations for each job
- \( p_{ij} \) processing times of operations \( O_{ij} \)
- \( \mu_{ij} \subseteq M \) machine eligibilities for each operation
- precedence constraints among the operations
- one job processed per machine at a time,
  one machine processing each job at a time
- \( C_j \) completion time of job \( j \)

Flow Shop

Find feasible schedule that minimize some regular function of \( C_j \)
Flow Shop

General Shop Scheduling:
- \( J = \{1, \ldots, N\} \) set of jobs; \( M = \{1, 2, \ldots, m\} \) set of machines
- \( J_j = \{O_{ij} \mid i = 1, \ldots, n_j\} \) set of operations for each job
- \( p_{ij} \) processing times of operations \( O_{ij} \)
- \( \mu_{ij} \subseteq M \) machine eligibilities for each operation
- precedence constraints among the operations
- one job processed per machine at a time, one machine processing each job at a time
- \( C_j \) completion time of job \( j \)
- Find feasible schedule that minimize some regular function of \( C_j \)

Example

<table>
<thead>
<tr>
<th>jobs</th>
<th>( j_1 )</th>
<th>( j_2 )</th>
<th>( j_3 )</th>
<th>( j_4 )</th>
<th>( j_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_{1,j_1} )</td>
<td>5 5 3 6 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( p_{2,j_1} )</td>
<td>4 4 2 4 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( p_{3,j_1} )</td>
<td>4 4 3 4 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( p_{4,j_1} )</td>
<td>3 6 3 2 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Flow Shop

General Shop Scheduling:
- \( J = \{1, \ldots, N\} \) set of jobs; \( M = \{1, 2, \ldots, m\} \) set of machines
- \( J_j = \{O_{ij} \mid i = 1, \ldots, n_j\} \) set of operations for each job
- \( p_{ij} \) processing times of operations \( O_{ij} \)
- \( \mu_{ij} \subseteq M \) machine eligibilities for each operation
- precedence constraints among the operations
- one job processed per machine at a time, one machine processing each job at a time
- \( C_j \) completion time of job \( j \)
- Find feasible schedule that minimize some regular function of \( C_j \)

Flow Shop Scheduling:
- \( \mu_{ij} = l, l = 1, 2, \ldots, m \)
- precedence constraints: \( O_{ij} \rightarrow O_{i+1,j}, i = 1, 2, \ldots, n \) for all jobs

Example

<table>
<thead>
<tr>
<th>jobs</th>
<th>( j_1 )</th>
<th>( j_2 )</th>
<th>( j_3 )</th>
<th>( j_4 )</th>
<th>( j_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_{1,j_2} )</td>
<td>5 5 3 6 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( p_{2,j_2} )</td>
<td>4 4 2 4 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( p_{3,j_2} )</td>
<td>4 4 3 4 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( p_{4,j_2} )</td>
<td>3 6 3 2 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

schedule representation \( \pi_1, \pi_2, \pi_3, \pi_4 \):
Example

<table>
<thead>
<tr>
<th>jobs</th>
<th>j₁</th>
<th>j₂</th>
<th>j₃</th>
<th>j₄</th>
<th>j₅</th>
</tr>
</thead>
<tbody>
<tr>
<td>p₁,jₖ</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>p₂,jₖ</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>p₃,j₆</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>p₄,j₆</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

schedule representation
π₁, π₂, π₃, π₄:
π₁ : O₁₁, O₁₂, O₁₃, O₁₄
π₂ : O₂₁, O₂₂, O₂₃, O₂₄
π₃ : O₃₁, O₃₂, O₃₃, O₃₄
π₄ : O₄₁, O₄₂, O₄₃, O₄₄

Example

<table>
<thead>
<tr>
<th>jobs</th>
<th>j₁</th>
<th>j₂</th>
<th>j₃</th>
<th>j₄</th>
<th>j₅</th>
</tr>
</thead>
<tbody>
<tr>
<td>p₁,jₖ</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>p₂,jₖ</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>p₃,j₆</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>p₄,j₆</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

schedule representation
π₁, π₂, π₃, π₄:
π₁ : O₁₁, O₁₂, O₁₃, O₁₄
π₂ : O₂₁, O₂₂, O₂₃, O₂₄
π₃ : O₃₁, O₃₂, O₃₃, O₃₄
π₄ : O₄₁, O₄₂, O₄₃, O₄₄

Outline

1. Parallel Machine Models
2. Flow Shop

Example

<table>
<thead>
<tr>
<th>jobs</th>
<th>j₁</th>
<th>j₂</th>
<th>j₃</th>
<th>j₄</th>
<th>j₅</th>
</tr>
</thead>
<tbody>
<tr>
<td>p₁,jₖ</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>p₂,jₖ</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>p₃,j₆</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>p₄,j₆</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

schedule representation
π₁, π₂, π₃, π₄:
π₁ : O₁₁, O₁₂, O₁₃, O₁₄
π₂ : O₂₁, O₂₂, O₂₃, O₂₄
π₃ : O₃₁, O₃₂, O₃₃, O₃₄
π₄ : O₄₁, O₄₂, O₄₃, O₄₄

- we assume unlimited buffer
- if same job sequence on each machine ➔ permutation flow shop
Directed Graph Representation

Given a sequence: operation-on-node network, jobs on columns, and machines on rows

Recursion for $C_{\text{max}}$

$$C_{i,\pi(1)} = \sum_{l=1}^{i} p_{l,\pi(1)}$$

$$C_{1,\pi(j)} = \sum_{l=1}^{j} p_{l,\pi(l)}$$

$$C_{i,\pi(j)} = \max\{C_{i-1,\pi(j)}, C_{i,\pi(j-1)}\} + p_{i,\pi(j)}$$

Computation cost?

Example

<table>
<thead>
<tr>
<th>jobs</th>
<th>$j_1$</th>
<th>$j_2$</th>
<th>$j_3$</th>
<th>$j_4$</th>
<th>$j_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_{1,j_1}$</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>$p_{2,j_2}$</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>$p_{3,j_3}$</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>$p_{4,j_4}$</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

$C_{\text{max}} = 34$
corresponds to longest path

Outline

1. Parallel Machine Models

2. Flow Shop
   - Introduction
   - Makespan calculation
   - Johnson’s algorithm
   - Construction heuristics
   - Iterated Greedy
   - Efficient Local Search and Tabu Search

**Theorem**

There always exist an optimum sequence without change in the first two and last two machines.

**Proof:** By contradiction.

\[
\begin{align*}
M_1 & \quad \cdots \quad i \quad l \quad \cdots \quad h \quad j \\
M_2 & \quad \cdots \quad i \quad j
\end{align*}
\]

**Corollary**

\( F_2 \mid \max C \) and \( F_3 \mid \max C \) are permutation flow shop

**Note:** \( F_3 \mid \max C \) is strongly NP-hard

**Intuition:**

give something short to process to 1 such that 2 becomes operative and give something long to process to 2 such that its buffer has time to fill.
**F2 | C_{max}**

**Intuition:** give something short to process to 1 such that 2 becomes operative and give something long to process to 2 such that its buffer has time to fill.

Constructs a sequence \( T : T(1), \ldots, T(n) \) to process in the same order on both machines by concatenating two sequences:
- a left sequence \( L : L(1), \ldots, L(t) \), and a right sequence \( R : R(t + 1), \ldots, R(n) \), that is, \( T = L \circ R \)

[Selmer Johnson, 1954, Naval Research Logistic Quarterly]

Let \( J \) be the set of jobs to process
Let \( T, L, R = \emptyset \)

**Step 1** Find \((i^*, j^*)\) such that \( p_{i^* \cdot j^*} = \min \{ p_{i,j} \mid i \in 1, 2, j \in J \} \)

---

**F2 | C_{max}**

**Intuition:** give something short to process to 1 such that 2 becomes operative and give something long to process to 2 such that its buffer has time to fill.

Constructs a sequence \( T : T(1), \ldots, T(n) \) to process in the same order on both machines by concatenating two sequences:
- a left sequence \( L : L(1), \ldots, L(t) \), and a right sequence \( R : R(t + 1), \ldots, R(n) \), that is, \( T = L \circ R \)

[Selmer Johnson, 1954, Naval Research Logistic Quarterly]

Let \( J \) be the set of jobs to process
Let \( T, L, R = \emptyset \)

**Step 1** Find \((i^*, j^*)\) such that \( p_{i^* \cdot j^*} = \min \{ p_{i,j} \mid i \in 1, 2, j \in J \} \)

**Step 2** If \( i^* = 1 \) then \( L = L \circ \{ i^* \} \)
else if \( i^* = 2 \) then \( R = R \circ \{ i^* \} \)
**F2** || \( C_{\text{max}} \)

**Intuition:** give something short to process to 1 such that 2 becomes operative and give something long to process to 2 such that its buffer has time to fill.

Constructs a sequence \( T : T(1), \ldots, T(n) \) to process in the same order on both machines by concatenating two sequences:
a left sequence \( L : L(1), \ldots, L(t) \), and a right sequence \( R : R(t+1), \ldots, R(n) \), that is, \( T = L \circ R \)

[Selmer Johnson, 1954, Naval Research Logistic Quarterly]

Let \( J \) be the set of jobs to process
Let \( T, L, R = \emptyset \)
Step 1 Find \((i^*, j^*)\) such that \( p_{i^*, j^*} = \min \{ p_{ij} | i \in 1, 2, j \in J \} \)
Step 2 If \( i^* = 1 \) then \( L = L \circ \{ i^* \} \)
     else if \( i^* = 2 \) then \( R = R \circ \{ i^* \} \)
Step 3 \( J := J \setminus \{ j^* \} \)
Step 4 If \( J \neq \emptyset \) go to Step 1 else \( T = L \circ R \)

**Theorem**

The sequence \( T : T(1), \ldots, T(n) \) is optimal.

**Proof**

- Assume at one iteration of the algorithm that job \( k \) has the min processing time on machine 1. Show that in this case job \( k \) has to go first on machine 1 than any other job selected later.

- By contradiction, show that if in a schedule \( S \) a job \( j \) precedes \( k \) on machine 1 and has larger processing time on 1, then \( S \) is a worse schedule than \( S' \).
- There are three cases to consider.
- Iterate the prove for all jobs in \( L \).
- Prove symmetrically for all jobs in \( R \).
Construction Heuristics (1)

Fm | prmu | Cmax

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 $m - 1$

\[ p'_{ij} = \sum_{k=1}^{i} p_{kj} \quad p''_{ij} = \sum_{k=m-i+1}^{m} p_{kj} \]

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

Construction Heuristics (2)

Fm | prmu | Cmax

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 $O(n^2m)$

Implementation in $O(n^2m)$

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

1. Parallel Machine Models

2. Flow Shop
   - Introduction
   - Makespan calculation
   - Johnson’s algorithm
   - Construction heuristics
     Iterated Greedy
     Efficient Local Search and Tabu Search

Iterated Greedy

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 \( O(n^2m) \))

**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.
Efficient local search for $Fm|prmu|C_{max}$

Tabu search (TS) with insert neighborhood.

TS uses best strategy. ⇔ need to search efficiently!

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

A sequence $t = (t_1, t_2, \ldots, t_{m-1})$ defines a path in $\pi$:

$$C(\pi, t) = \sum_{j=1}^{n} p_{n(j)1} + \sum_{j=t_1}^{t_2} p_{n(j)2} + \cdots + \sum_{j=t_1}^{t_{m-1}} p_{n(j)m}$$

$C_{max}$ expression through critical path:

$$C_{max}(\pi) = \max_{1 \leq t_1 \leq t_2 \leq \cdots \leq t_{m-1} \leq n} \left( \sum_{j=1}^{n} p_{n(j)1} + \sum_{j=t_1}^{t_2} p_{n(j)2} + \cdots + \sum_{j=t_1}^{t_{m-1}} p_{n(j)m} \right)$$
critical path: $\vec{u} = (u_1, u_2, \ldots, u_m): C_{\max}(\pi) = C(\pi, u)$

Block $B_k$ and Internal Block $B_{k}^{\text{Int}}$

Corollary (Elimination Criterion)
If $\pi'$ is obtained by $\pi$ by an "internal block insertion" then $C_{\max}(\pi') \geq C_{\max}(\pi)$.

Theorem (Werner, 1992)
Let $\pi, \pi' \in \Pi$, if $\pi'$ has been obtained from $\pi$ by an job insert so that $C_{\max}(\pi') < C_{\max}(\pi)$ then in $\pi'$:

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 $\pi(u_k)$, $k = 1, \ldots, m$
Corollary (Elimination Criterion)

If \( \pi' \) is obtained by \( \pi \) by an “internal block insertion” then

\[
C_{\text{max}}(\pi') \geq C_{\text{max}}(\pi).
\]

Hence we can restrict the search to where the good moves can be:

Further speedup: Use of lower bounds in delta evaluations:
Let \( \delta_{x,u_k} \) indicate insertion of \( x \) after \( u_k \) (move of type \( ZR_k(\pi) \))

\[
\Delta(\delta_{x,u_k}) = \begin{cases} P\pi(x,k+1) - P\pi(u_k),k+1 & x \neq u_k-1 \\ P\pi(x,k+1) - P\pi(u_k),k+1 + P\pi(u_{k-1},k-1) - P\pi(x),k-1 & x = u_k-1 \end{cases}
\]

That is, add and remove from the adjacent blocks

It can be shown that:

\[
C_{\text{max}}(\delta_{x,u_k}(\pi)) \geq C_{\text{max}}(\pi) + \Delta(\delta_{x,u_k})
\]

Further speedup: Use of lower bounds in delta evaluations:
Let \( \delta_{x,u_k} \) indicate insertion of \( x \) after \( u_k \) (move of type \( ZR_k(\pi) \))

\[
\Delta(\delta_{x,u_k}) = \begin{cases} P\pi(x,k+1) - P\pi(u_k),k+1 & x \neq u_k-1 \\ P\pi(x,k+1) - P\pi(u_k),k+1 + P\pi(u_{k-1},k-1) - P\pi(x),k-1 & x = u_k-1 \end{cases}
\]

That is, add and remove from the adjacent blocks

It can be shown that:

\[
C_{\text{max}}(\delta_{x,u_k}(\pi)) \geq C_{\text{max}}(\pi) + \Delta(\delta_{x,u_k})
\]

Theorem (Nowicki and Smutnicki, 1996, EJOR)

The neighborhood thus defined is connected.
Metaheuristic details:

Prohibition criterion:
an insertion $\delta_{x,u_k}$ is tabu if it restores the relative order of $\pi(x)$ and $\pi(x+1)$.

Tabu length: $TL = 6 + \left\lceil \frac{n}{10m} \right\rceil$

Perturbation

- perform all inserts among all the blocks that have $\Delta < 0$
- 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 $\Delta$.
Apply move. Compute true $C_{max}(\delta(\pi))$.
If improving compare with $C^*$ and in case update. Else increase number of idle iterations.

Perturbation: Apply perturbation if MaxIdleIter done.
Stop criterion: Exit if MaxIter iterations are done.