Resume

Job Shop:
- Definition
- Starting times and \( m \)-tuple permutation representation
- Disjunctive graph representation [Roy and Sussman, 1964]
- Shifting Bottleneck Heuristic [Adams, Balas and Zawack, 1988]
Job Shop Generalizations

Generalizations: Time Lags

Generalized time constraints

They can be used to model:

- **Release time:**
  \[ S_0 + r_i \leq S_i \iff d_{0i} = r_i \]

- **Deadlines:**
  \[ S_i + p_i - d_i \leq S_0 \iff d_{i0} = p_i - d_i \]

**Modelling**

\[
\begin{align*}
\min & \quad C_{\max} \\
\text{s.t.} & \quad x_{ij} + d_{ij} \leq C_{\max} \quad \forall O_{ij} \in N \\
& \quad x_{ij} + d_{ij} \leq x_{ij} \quad \forall (O_{ij}, O_{ij}) \in A \\
& \quad x_{ij} + d_{ij} \leq x_{ik} \lor x_{ij} + d_{ij} \leq x_{ik} \quad \forall (O_{ij}, O_{ik}) \in E \\
& \quad x_{ij} \geq 0 \\
& \quad i = 1, \ldots, m \quad j = 1, \ldots, N
\end{align*}
\]

- **Exact relative timing (perishability constraints):**
  if operation \( j \) must start \( l_{ij} \) after operation \( i \):
  \[ S_i + p_i + l_{ij} \leq S_j \quad \text{and} \quad S_j - (p_i + l_{ij}) \leq S_i \]
  \((l_{ij} = 0 \text{ if no-wait constraint})\)

Set up times:

\[ S_i + p_i + s_{ij} \leq S_j \quad \text{or} \quad S_j + s_{ij} \leq S_i \]

**Machine unavailabilities:**

- Machine \( M_k \) unavailable in \([a_1, b_1], [a_2, b_2], \ldots, [a_v, b_v]\)
- Introduce \( v \) artificial operations with \( \lambda = 1, \ldots, v \) with \( \mu_\lambda = M_k \) and:
  \[ p_\lambda = b_\lambda - a_\lambda \]
  \[ r_\lambda = a_\lambda \]
  \[ d_\lambda = b_\lambda \]

**Minimum lateness objectives:**

\[ L_{\max} = \max_{j=1}^{N}(C_j - d_{j}) \iff d_{n_j,n+1} = p_{n_j} - d_{j} \]
Arises with limited buffers:
after processing, a job remains on the machine until the next machine is freed

- Needed a generalization of the disjunctive graph model
  \[ \rightarrow \text{Alternative graph model} \ G = (N, E, A) \] [Mascis, Pacciarelli, 2002]

1. Two non-blocking operations to be processed on the same machine
   \[ S_i + p_i \leq S_j \quad \text{or} \quad S_j + p_j \leq S_i \]

2. Two blocking operations \( i, j \) to be processed on the same machine \( \mu(i) = \mu(j) \)
   \[ S_{\sigma(j)} \leq S_i \quad \text{or} \quad S_{\sigma(i)} \leq S_j \]

3. \( i \) is blocking, \( j \) is non-blocking (ideal) and \( i, j \) to be processed on the same machine
   \[ \mu(i) = \mu(j). \]
   \[ S_i + p_i \leq S_j \quad \text{or} \quad S_{\sigma(j)} \leq S_i \]

- A complete selection \( S \) is consistent if it chooses alternatives from each pair such that the resulting graph does not contain positive cycles.

Example:

- \( M(O_1) = M(O_5) = M(O_9) \)
- \( M(O_2) = M(O_6) = M(O_{10}) \)
- \( M(O_3) = M(O_7) = M(O_{11}) \)

Example:

- \( p_a = 4 \)
- \( p_b = 2 \)
- \( p_c = 1 \)
- \( b \) must start at least 9 days after \( a \) has started
- \( c \) must start at least 8 days after \( b \) is finished
- \( c \) must finish within 16 days after \( a \) has started

\[ S_a + 9 \leq S_b \]
\[ S_b + 10 \leq S_c \]
\[ S_c - 15 \leq S_a \]

This leads to an absurd.
In the alternative graph the cycle is positive.
The Makespan still corresponds to the longest path in the graph with the arc selection $G(S)$.

Problem: now the digraph may contain cycles. Longest path with simple cyclic paths is NP-complete. However, here we have to care only of non-positive cycles.

If there are no cycles of length strictly positive it can still be computed efficiently in $O(|N| |E \cup A|)$ by Bellman-Ford (1958) algorithm.

The algorithm iteratively considers all edges in a certain order and updates an array of longest path lengths for each vertex. It stops if a loop over all edges does not yield any update or after $|N|$ iterations over all edges (in which case we know there is a positive cycle).

Possible to maintain incremental updates when changing the selection [Demetrescu, Frangioni, Marchetti-Spaccamela, Nanni, 2000].

**Rollout**

- **Master process:** grows a partial selection $S^k$:
  decides the next element to fix based on a heuristic function (selects the one with minimal value)

- **Slave process:** evaluates heuristically the alternative choices.
  Completes the selection by keeping fixed what passed by the master process and fixing one alternative at a time.

**Slave heuristics**

- **Avoid Maximum Current Completion time**
  find an arc $(h,k)$ that if selected would increase most the length of the longest path in $G(S^k)$ and select its alternative
  $$\max_{(uv) \in A} \{l(0,u) + a_{uv} + l(u,n)\}$$

- **Select Most Critical Pair**
  find the pair that, in the worst case, would increase least the length of the longest path in $G(S^k)$ and select the best alternative
  $$\max_{((ij),(hk)) \in A} \min\{l(0,u) + a_{hk} + l(k,n), l(0, i) + a_{ij} + l(j,n)\}$$

- **Select Max Sum Pair**
  finds the pair with greatest potential effect on the length of the longest path in $G(S^k)$ and select the best alternative
  $$\max_{((ij),(hk)) \in A} |l(0,u) + a_{hk} + l(k,n) + l(0, i) + a_{ij} + l(j,n)|$$

Trade off quality vs keeping feasibility

Results depend on the characteristics of the instance.

- The search space is highly constrained + detecting positive cycles is costly

- Hence local search methods not very successful

- Rely on the construction paradigm

- **Rollout algorithm** [Meloni, Pacciarelli, Pranzo, 2004]
Implementation details of the slave heuristics

- Once an arc is added we need to update all $L(0, u)$ and $L(u, n)$. Backward and forward visit $O(|F| + |A|)$

- When adding arc $a_{ij}$, we detect positive cycles if $L(i, j) + a_{ij} > 0$. This happens only if we updated $L(0, i)$ or $L(j, n)$ in the previous point and hence it comes for free.

- Overall complexity $O(|A|(|F| + |A|))$

Speed up of Rollout:

- Stop if partial solution overtakes upper bound

- limit evaluation to say 20% of arcs in $A$