Notes on Advanced Optimization

40.302 Advanced Topics in Optimization, Term 7 2019

Yustynn Panicker

March 6, 2019

Contents

1 Meta **5**
- 1.1 Professor . . . 5
- 1.2 Unofficial Project . . . 5
- 1.3 Grading . . . 5
- 1.4 Credits . . . 5

2 L1: Introduction to Scheduling **5**
- 2.1 Sugested Language . . . 5
- 2.2 Why Scheduling? . . . 5
- 2.3 Definition of Scheduling . . . 5
  - 2.3.1 Assumptions . . . 5
- 2.4 Terms and Notation . . . 6
  - 2.4.1 General . . . 6
  - 2.4.2 Jobs . . . 6
  - 2.4.3 Machine Environment (α) . . . 6
    - 2.4.3.1 Hierarchical Representation . . . 7
  - 2.4.4 **TODO** Processing Details and Constraints (β) . . . 7
  - 2.4.5 Objective Criteria (γ) . . . 7
    - 2.4.5.1 Hierarchical Representation . . . 8
  - 2.4.6 Terms . . . 8
    - 2.4.6.1 Forced Idleness . . . 8
    - 2.4.6.2 Irregular functions . . . 8
    - 2.4.6.3 Active Schedule . . . 9
- 2.5 Theorems . . . 9
  - 2.5.1 Theorem 1 . . . 9
    - 2.5.1.1 Proof Sketch . . . 9
      - 2.5.1.1.1 Counter . . . 9
      - 2.5.1.1.2 Procedure Definition . . . 9
      - 2.5.1.1.3 Proof of Optimality . . . 9
      - 2.5.1.1.4 Iteration and Limit of Iteration . . . 9
      - 2.5.1.1.5 Conclusion . . . 9
  - 2.5.2 Theorem 2 . . . 9
- 2.6 Anomalies in non-preemptive non-delay schedules . . . 9
  - 2.6.1 Example . . . 10
    - 2.6.1.1 Initial . . . 10
    - 2.6.1.2 Assuming Job 7 is shorter . . . 10
    - 2.6.1.3 (Invalid) Better Solution . . . 10

3 L2: Single Machine Scheduling **11**
3.1 General Solutions . . . 11
3.1.1 Deterministic . . . 11
3.1.2 Heuristics . . . 11
3.1.2.1 $1|r_j|\sum_j C_j$ . . . 11
3.1.2.1.1 Heuristic A . . . 11
3.1.2.1.2 Heuristic B . . . 11
3.2 Proofs by Adjacent Pairwise Arguments . . . 12
3.2.1 SPT for $1||\sum_j C_j$ . . . 12
3.2.2 WSPT for $1||\sum_j w_j C_j$ . . . 13
3.3 Integer Program Formulations . . . 13
3.3.1 Selin's Notation . . . 13
3.3.1.1 Time-Indexed Variables (nt variables) . . . 13
3.3.1.2 Linear Ordering (Precedence) Variables ($n(n-1)$ variables) . . . 13
3.3.1.3 Sequence-Position (rank) Variables (n^2 variables) . . . 13
3.3.2 $1|r_j|\sum_j w_j C_j$ . . . 14
3.3.2.1 Using linear ordering variables . . . 14
3.3.2.2 Using time indexed variables . . . 14
3.3.2.3 Using sequence-position variables . . . 15
3.3.2.3.1 Explanatory notes . . . 15
3.3.3 $1|\text{prec}|\sum_j w_j C_j$ . . . 15
3.4 Chains . . . 15
3.4.1 $1|\text{chain}|\sum_j w_j C_j$ . . . 15
3.4.1.1 ρ-factor and l^* . . . 15
3.4.1.2 Lemma: On choosing which sequence to process first . . . 16
3.4.1.2.1 Proof . . . 16
3.4.1.3 Lemma: Process up to l^* without interruption . . . 16
3.4.1.3.1 Proof . . . 16
3.4.1.4 Algorithm . . . 17
3.5 Incurred Costs (Function $h_j(C)$) . . . 17
3.5.1 $1|prec|h_{\max}$ . . . 17
3.5.1.1 Algorithm . . . 17
3.5.1.2 (Brief) Proof Sketch of Scheduling k in last available slot . . . 18
3.5.2 Dynamic Programming for $1||\sum_j h_j(C_j)$ . . . 18
3.5.2.1 Time Complexity Savings . . . 18
3.5.2.2 Why it Works . . . 18
3.5.2.3 Example . . . 19
3.6 Misc . . . 19
3.6.1 Relationship between average completion times and inventory costs . . . 19
3.7 Modelling Strategies . . . 19
3.7.1 Disjunctive Constraints (XOR) . . . 19
3.7.1.1 Description . . . 19
3.7.1.1.1 E.g. of disconnected feasible region shown below. . . . 20
3.7.1.2 Modelling Strategy: Big M . . . 20
3.7.1.2.1 Example Problem . . . 20
3.7.1.2.2 Example Big M Usage . . . 20

4 L3: $C_{\max}$ for Parallel Machine Environments **21**
4.1 $C_{\max}$ lower bounds . . . 21
4.2 $P_m|prmp|C_{\max}$ . . . 21
4.2.1 Integer Program Formulation . . . 21
4.2.2 Algorithm . . . 22
4.3 $P_m||C_{\max}$ (no preemption) . . . 22

4.3.1 Partition Problem . . . 22
4.3.1.1 Informal explanation . . . 22
4.3.1.2 Formal . . . 22
4.3.1.3 Relation to $P_2||C_{\max}$ . . . 22
4.3.2 Integer Program Formulation . . . 22
4.3.3 List Scheduling Algorithm (Heuristic) . . . 23
4.3.3.1 Algorithm . . . 23
4.3.3.2 ρ-approximation . . . 23
4.3.3.3 Proof of ρ-approximation . . . 23

5 L4: Flow Shop Scheduling **24**
5.1 Terminology . . . 24
5.2 Misc . . . 24
5.3 Permutation Problem $F_m|prmu|C_{\max}$ . . . 24
5.3.1 Graphical Interpretation . . . 24
5.3.2 Exploitable Knowledge . . . 24
5.3.3 Recursive Solution . . . 25
5.3.3.1 Explanation of recursive formulation . . . 25
5.4 No Permutation $F||C_{\max}$ . . . 25
5.4.1 Integer Programming . . . 25
5.4.1.1 Strategy Comments . . . 25
5.4.1.2 Relationship between $I_{i,k}$ and $W_{i,k}$ . . . 25
5.4.1.3 Idea for alternate formulation of objective function . . . 26
5.4.1.4 Naive Objective Function . . . 26
5.4.1.5 Full Formulation . . . 27
5.4.2 Theorem: Sufficient to consider schedules which maintain processing order (2-3 machines) . 27
5.4.2.1 Intuition and Proof Sketch . . . 27
5.4.2.2 Why this works for 3 machines . . . 27
5.4.3 Idle Time Theorem for 2 machines . . . 28
5.4.3.1 Corollary 1 . . . 28
5.4.3.2 Corollary 2 . . . 28
5.4.4 Johnson's Rule . . . 28
5.4.4.1 Algorithm . . . 28
5.4.4.2 Notes . . . 28
5.4.4.3 Example . . . 29
5.4.4.4 Proof . . . 29
5.4.4.4.1 Comments . . . 29

6 TODO L5: Job Shop Scheduling **30**
6.1 Disjunctive Graph Formulation . . . 30
6.2 **TODO** Integer Program Formulation (Based on Disjunctive Graph) . . . 30

7 TODO L6: Uncertainty in Scheduling (Stochastic) **30**
7.1 General . . . 30
7.2 Potential Areas of Randomness . . . 30

8 Proving Techniques **30**
8.1 Induction . . . 30
8.2 Enumeration/Construction/Direct (brute force) . . . 31
8.3 Contradiction (very useful!) . . . 31
8.4 Contra-positive . . . 31

9 TODO To Revise **32**
9.1 Newsvendor . . . 32

9.2 Dynamic Programming . . . 32
9.2.1 Components . . . 32
9.2.2 Example . . . 32
9.2.3 DP-tacklable problems . . . 32
9.3 **TODO** Simplex . . . 32
9.4 **TODO** Dijkstra . . . 32
9.5 **TODO** Branch and Bound . . . 32

10 Misc **32**
10.0.1 Story . . . 32
10.0.2 Formulation . . . 32
10.0.3 Optimal Solution . . . 33

11 AMPL **33**
11.1 General . . . 33
11.2 Tips . . . 33

1 Meta

1.1 Professor

- Selin Damla Ahipasaoglu
- Room 1.702.03

1.2 Unofficial Project

Nothing official this term due to large class size. Can still do, and approach Selin for ideas

1.3 Grading

%	Component Name	Date
60%	Final Exam	W6
20%	Weekly Homeworks	Assigned Mon; Due next Thurs;
20%	Class and Lab Participation	-

1.4 Credits

Significant portions of these notes are just copy-pasted segments of Selin's lecture slides

2 L1: Introduction to Scheduling

2.1 Sugested Language

AMPL (A Mathematical Programming Language)

2.2 Why Scheduling?

- Very hard problem -> Good practice for opti/modelling

2.3 Definition of Scheduling

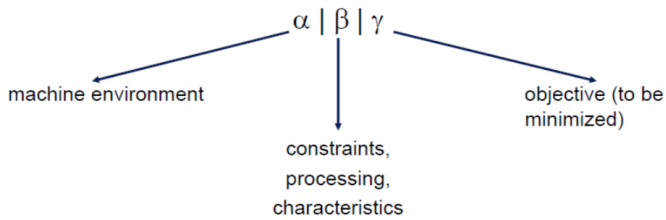
Deals with distributing a set of **jobs (tasks)** to **machines (processing units) over *time** subject to **constraints** with an objective to optimize ≥ 1 criteria

2.3.1 Assumptions

- Each machine has capacity of 1 job
- At each time unit, each job can be processed only on one machine

2.4 Terms and Notation

2.4.1 General



E.g. $J_m \mid\mid C_{max}$

Symbol	Meaning
$M = \{1, 2, \ldots, m\}$	Set of machines
$N = \{1, 2, \ldots, n\}$	Set of jobs

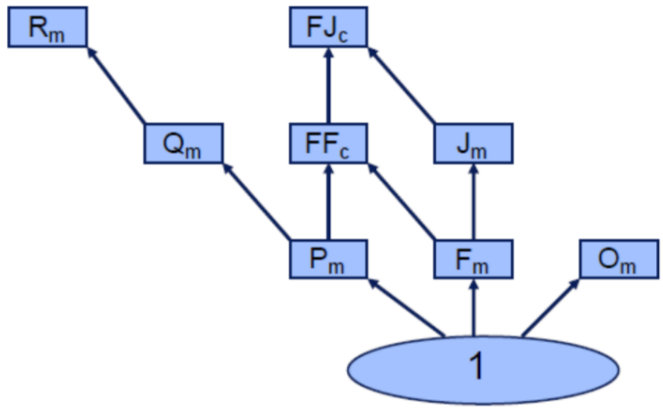
2.4.2 Jobs

Symbol	Meaning
p_{ij}	`Processing time` of job j on machine i
p_j	`Identical processing time` of job j on all machines
r_j	`Release date` of job j
d_j	`Due date` of job j; Penalty after d_j
$\bar{d}_j$	`Deadline` of job j; Completion after $\bar{d}_j$ not allowed
w_j	`Weight` of job j

2.4.3 Machine Environment (α)

Symbol	Term	Description/notes
1	Single machine	
P_m	m *identical* machines in parallel	Job j requires single operation, may be processed on any of the m machines
Q_m	m *uniform* machines in parallel with different speeds	Speed of machine i is v_i; $p_{ij} = \frac{p_j}{v_i}$
R_m	m unrelated machines in parallel	Speed of machine i is v_i; $p_{ij} = \frac{p_j}{v_i}$
F_m	`Flow shop` with m machines in series	Each job must be processed on every machine; predetermined route
FF_c	`Flexible flow shop` with c stages in series	Several identical machines per stage; Each job can only be processed on 1 machine per stage
J_m	`Job shop` with m machines	Jobs have predetermined routes
FJ_c	`Flexible job shop` with c stages	Multiple identical machines per stage
O_m	`Open shop` with m machines	Each job must be processed on every machine; No predetermined routes

2.4.3.1 Hierarchical Representation



2.4.4 TODO Processing Details and Constraints (β)

@TODO permutation prmu

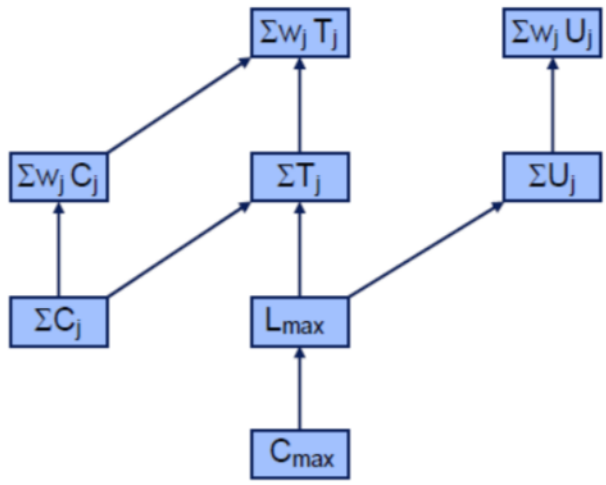
Notation	Definition/Explanation	Notes
r_j	Release dates given	
prmp	`Preemption` allowed (fractional processing)	Linear programming, $\therefore$ way easier (opposed to IP w/o *prmp*)
prmu	`Permutation` problem. Assume each machine follows FIFO	Makes it such that the processing order is the same on all machines
s_{jk}	`Sequence dependent setup times`	Setup times between processing jobs j and k; s_{ijk} has i refer to machine
prec	`Precedence constraints`	Usually represented with a directed acyclical graph (DAG)

Note: Due dates are reflected in objective function

2.4.5 Objective Criteria (γ)

Notation/Definition	Explanation	Notes
C_j	`Completion time` of job j	
$L_j := C_j - d_j$	`Lateness` of job j	+, - or 0
$T_j := max(C_j, 0)$	`Tardiness` of job j	+ or 0 only
$U_j = \begin{cases} 1, & \text{if } C_j > d_j \\ 0, & \text{o.w.} \end{cases}$	`Unit penalty` of job b being late	

2.4.5.1 Hierarchical Representation



E.g. L_{max} solver can be used for C_{max} by setting all $d_j = 0$

2.4.6 Terms

Term	Definition
recirculation	A machine is visited multiple times by the same job
makespan	Time of completion of last job
unforced idleness	Machine not processing anything; Jobs are availbale to be processed by it
forced idleness	Machine not processing anything; No jobs available to be processed by it
non-delay	No unforced idleness
regular objective function	Nondecreasing function when new input of $C' \geq C$ (as in, $C'_1 > C_1, \ldots, C'_n > C_n$
sequence	Of a specific machine. Refers to job order
schedule	Particular allocation of jobs to machines at times
scheduling policy	In stochastic settings, a prescription of appropriate action for a given state

2.4.6.1 Forced Idleness

Forced idleness can occur due to:

- precedence relationships
- release dates

2.4.6.2 Irregular functions

- Penalizing earliness as well as lateness
- Negative weights

2.4.6.3 Active Schedule

Prereqs:

- Feasible
- Nonpreemptive

Meaning: Not possible to construct another schedule where ≥ 1 operation finishes earlier and no operation finishes later

Basically, no fillable holes in schedule

2.5 Theorems

2.5.1 Theorem 1

For: $\alpha|prmp, \beta|\gamma$

Additional Constraints: γ is regular

Claim: There exists a `non-delay` schedule which is optimal

2.5.1.1 Proof Sketch

2.5.1.1.1 Counter Assume that *all* optimal schedules are *not* non-delay.

2.5.1.1.2 Procedure Definition Let S^1 be such an optimal schedule

Since there is unforced idleness, I can obtain another schedule S^2 by removing a complete segment of the unforced idleness due to preemption from S^1, by filling that idleness with ≥ 1 job

I can thus obtain another schedule S^2 such that $C_j^{S_1} \geq C_j^{S_2}, \forall j$, and where some $\mathrm{C}_\mathrm{j}^{\mathrm{S}_1} > \mathrm{C}_\mathrm{j}^{\mathrm{S}_2}$

2.5.1.1.3 Proof of Optimality Due to regularity, this means that $\gamma(C_J^{S_1}) \geq \gamma(C_J^{S_2})$. This implies S^2 is also optimal

2.5.1.1.4 Iteration and Limit of Iteration From this schedule, I can create another schedule S^3, and from that S^4 and so on until I hit a non-delay schedule (the limit of the iterative performance of this operation).

2.5.1.1.5 Conclusion This non-delay schedule is therefore optimal.

2.5.2 Theorem 2

For: $J_m||\gamma$

Additional Constraints: γ is regular

Claim: There exists an `active` schedule which is also optimal

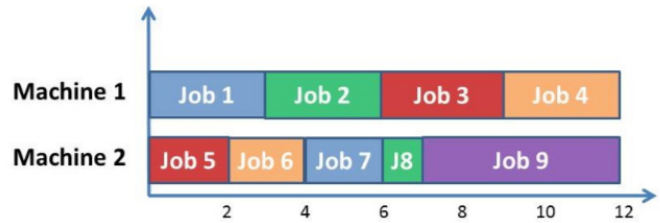
2.6 Anomalies in non-preemptive non-delay schedules

E.g. decreasing processing time may make objective function worse (since you can't delay)

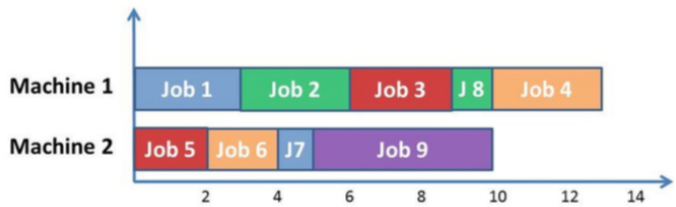
2.6.1 Example

2.6.1.1 Initial

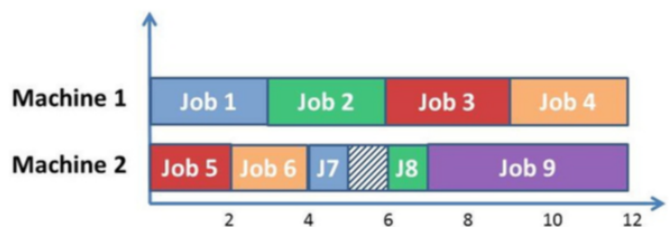
j	1	2	3	4	5	6	7	8	9
p_j	3	3	3	3	2	2	2	1	5
prec_j	-	1	2	3,8	-	5	6	2	7



2.6.1.2 Assuming Job 7 is shorter



2.6.1.3 (Invalid) Better Solution



Invalid due to delay

3 L2: Single Machine Scheduling

3.1 General Solutions

3.1.1 Deterministic

Formulation	Solution	Remarks
$1\|\|\sum_j C_j$	SPT (shortest processing time)	Sort in non-decreasing order
$1\|\|\sum_j w_j C_j$	WSPT aka Smith's rule (weighted shortest processing time)	$\frac{w_j}{p_j}$; sort in non-increasing order

It's weird that the WSPT rule is sorted in non-increasing order, while the SPT rule is sorted in non-decreasing order, but whatever. Invert the WSPT formula if it makes you feel better.

3.1.2 Heuristics

3.1.2.1 $1|r_j|\sum_j C_j$

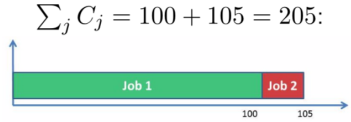
3.1.2.1.1 Heuristic A

- SPT rule among all jobs available at each given time
- Implies non-delay schedule. Note that optimal may be a delay schedule

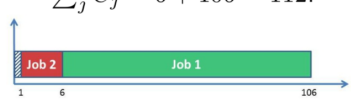
1. Weakness
   - Delaying a little to wait for a job with smaller p may be better
2. Example

j	1	2
p_j	100	5
r_j	0	1

Heuristic schedule has $\sum_j C_j = 100 + 105 = 205$:



Optimal schedule has $\sum_j C_j = 6 + 106 = 112$:



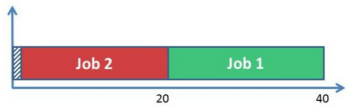
3.1.2.1.2 Heuristic B

- SPT rule, ignoring release dates
- Implies possible delay schedules (if $t < r_j$ for next job j).

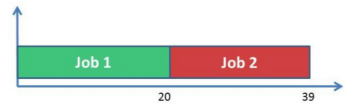
1. Weakness
   - Long unforced idleness can ensue
2. Example

j	1	2
p_j	20	19
r_j	0	1

Heuristic schedule has $\sum_j C_j = 20 + 40 = 60$:



Optimal schedule has $\sum_j C_j = 20 + 39 = 59$:

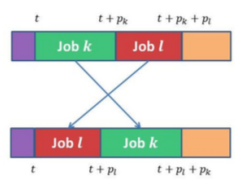


3.2 Proofs by Adjacent Pairwise Arguments

3.2.1 SPT for $1||\sum_j C_j$

Theorem: The SPT rule is optimal for $1||\sum_j C_j$.

Proof: By contradiction.
Assume that an optimal schedule S does not obey the SPT rule. Then there exists jobs k and l such that $p_k > p_l$ and job k is processed before job l in schedule S.



Consider a new schedule S' which is the same as S except the order of jobs k and l are swapped. The completion times of all jobs $j \neq k, l$ are unchanged in S and S'.

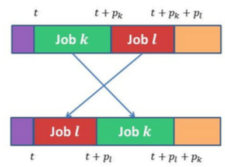
Proof: By contradiction.

$$\begin{aligned}
\gamma(S') - \gamma(S) &= \sum_j C_j^{S'} - \sum_j C_j^S = \sum_j (C_j^{S'} - C_j^S) \\
&= (t + p_l + t + p_l + p_k) - (t + p_k + t + p_k + p_l) \\
&= p_l - p_k < 0 \text{ (by assumption)}
\end{aligned}$$

This contradicts with the assumption that S is optimal.

3.2.2 WSPT for $1||\sum_j w_j C_j$

Proof: By contradiction.
Assume that an optimal schedule S does not obey the WSPT rule. Then there exists jobs k and l such that $\frac{w_k}{p_k} < \frac{w_l}{p_l}$ and job k is processed before job l in schedule S.



Consider a new schedule S' which is the same as S except the order of jobs k and l are swapped. The completion times of all jobs $j \neq k, l$ are unchanged in S and S'.

$$\gamma(S') - \gamma(S) = \sum_j w_j C_j^{S'} - \sum_j w_j C_j^S = \sum_j (w_j C_j^{S'} - w_j C_j^S)$$
$$= [w_l(t+p_l) + w_k(t+p_l+p_k)] - [w_k(t+p_k) + w_l(t+p_k+p_l)]$$
$$= w_k p_l - w_l p_k < 0 \text{ (by assumption)}$$

This contradicts with the assumption that S is optimal.

3.3 Integer Program Formulations

3.3.1 Selin's Notation

The important thing for each of these is retrieving C_j. The rest, you can just figure out on your own. Regardless, the full formulations are listed.

3.3.1.1 Time-Indexed Variables (nt variables)

$$X_{jt} = \begin{cases} 1 & \text{, if job } j \text{ starts at time } t \\ 0 & \text{, otherwise} \end{cases}$$

3.3.1.2 Linear Ordering (Precedence) Variables ($n(n-1)$ variables)

$$X_{jk} = \begin{cases} 1 & \text{, if job } j \text{ precedes job } k \\ 0 & \text{, otherwise} \end{cases}$$

3.3.1.3 Sequence-Position (rank) Variables (n^2 variables)

$$X_{jk} = \begin{cases} 1 & \text{, if job } j \text{ is assigned to position } k \\ 0 & \text{, otherwise} \end{cases}$$

3.3.2 $1|r_j|\sum_j w_j C_j$

3.3.2.1 Using linear ordering variables

$$
\begin{aligned}
\min \sum_j w_j C_j & \\
\text{s.t. } & C_j \geq p_j + r_j, && \forall j, (j_j \text{ only completed after it is released AND processed}) \\
& C_j + M X_{jk} \geq C_k + p_j, && \forall j < (\text{Either } j_j \text{ is processed after } j_k) \\
& C_k + M(1 - X_{jk}) \geq C_j + p_k, && \forall j < k, (\text{or the other way around}) \\
& X_{jk} \in \{0, 1\} && \forall j < k,
\end{aligned}
$$

We may choose M to be $M = \sum_j p_j + \max_{j=1,\ldots,n} r_j$

3.3.2.2 Using time indexed variables

$$
\begin{aligned}
\min \sum_{j=1}^{n} w_j C_j & \\
\text{s.t. } & C_j = \sum_{t=0}^{T} (t + p_j) X_{jt} && \forall j, (\text{Define } C_j) \\
& \sum_{t=0}^{T} X_{jt} = 1 && \forall j, (\text{Each job starts exactly once}) \\
& \sum_{j=1}^{n} \sum_{s=\max(t+1-p_j,0)}^{t} X_{js} \leq 1 && \forall t = 0, \ldots, T-1, (\text{Each job starts exactly once}) \\
& X_{jt} = 0 && \forall t < r_j,\ j = 1, \ldots, n \\
& X_{jt} \in \{0, 1\} && \forall j = 0, \ldots, n,\ \forall t = 0, \ldots, T
\end{aligned}
$$

3.3.2.3 Using sequence-position variables

$$
\begin{aligned}
\min \sum_{j=1}^{n} w_j C_j & \\
\text{s.t. } & \sum_{k=1}^{n} X_{jk} = 1 && \forall j, \text{(Each job is processed at exactly one position)} \\
& \sum_{j=1}^{n} X_{jk} = 1 && \forall j, \text{(Each position has exactly one job)} \\
& \hat{C}_k \geq \sum_{j=1}^{n} (p_j + r_j) X_{jk} && \forall j, \text{(Jobs can't start until after their } r\text{)} \\
& \hat{C}_k \geq \hat{C}_{k-1} + \sum_{j=1}^{n} p_j X_{jk} && \forall k, \text{(Jobs at position } k \text{ can't start until } k-1 \text{ is done)} \\
& C_j \geq \hat{C}_k - M(1 - X_{jk}), \quad \forall j = 1, \ldots, n, \ \forall k = 1, \ldots, n \text{(Only one } \hat{C}_k \text{ corresponds to } C_j\text{)} \\
& C_j \geq 0 && \forall j = 1, \ldots, n \\
& X_{jk} \in \{0, 1\} && \forall j = 1, \ldots, n \ \forall k = 1, \ldots, n
\end{aligned}
$$

3.3.2.3.1 Explanatory notes

- $\hat{C}_k = C_{\sigma(k)}$
- The real trick here is the second last set of C_j constraints, which obtain the C_j lower bound from the different $\hat{C}_k$. Pay attention to it.

3.3.3 $1|\textbf{prec}|\sum_j w_j C_j$

$$
\begin{aligned}
\min \sum_{j} w_j C_j & \\
\text{s.t } & C_j \geq p_j + r_j && \forall j = 1, \ldots, n \text{(} j_j \text{ completed only after processing. Processing completed} \\
& C_k \geq C_j + p_k && \forall j < k, \text{ with } j \to k \text{(precedence relationships)} \\
& C_j + M X_{jk} \geq C_k + p_j, && \forall j < k, \text{(Either } j_j \text{ is processed after } j_k\text{)} \\
& C_k + M(1 - X_{jk}) \geq C_j + p_k, && \forall j < k, \text{(or the other way around)} \\
& X_{jk} \in \{0, 1\} && \forall j < k,
\end{aligned}
$$

3.4 Chains

3.4.1 $1|\textbf{chain}|\sum_j w_j C_j$

3.4.1.1 ρ-factor and l^*

$$
\rho(1, \ldots, k) = \max_{l=1,\ldots,k} \frac{\sum_{j=1}^{l} w_j}{\sum_{j=1}^{l} p_j} = \frac{\sum_{j=1}^{l^*} w_j}{\sum_{j=1}^{l^*} p_j}
$$

3.4.1.2 Lemma: On choosing which sequence to process first

$\frac{\sum_{j=1}^{k} w_j}{\sum_{j=1}^{k} p_j} > (<) \frac{\sum_{j=k+1}^{n} w_j}{\sum_{j=k+1}^{n} p_j} \iff$ it is optimal to process chain of jobs $1, \ldots, k$ before (after) chain of jobs k + 1, $\ldots, n$

3.4.1.2.1 Proof

$$\sum_j w_j C_j^S = \sum_{j=1}^{k} w_j (\sum_{l=1}^{j} p_l) + \sum_{j=k+1}^{n} w_j (\sum_{l=1}^{j} p_l).$$

For sequence $k+1, \ldots, n, 1, 2, \ldots, k$, weighted completion time is

$$\sum_j w_j C_j^{S'} = \sum_{j=k+1}^{n} w_j (\sum_{l=k+1}^{j} p_l) + \sum_{j=1}^{k} w_j (\sum_{l=k+1}^{n} p_l + \sum_{l=1}^{j} p_l).$$

$$\sum_j w_j C_j^S < \sum_j w_j C_j^{S'} \iff (\sum_{j=1}^{k} w_j)(\sum_{j=k+1}^{n} p_j) > (\sum_{j=k+1}^{n} w_j)(\sum_{j=1}^{k} p_j).$$

The results follows.

3.4.1.3 Lemma: Process up to l^* without interruption

Process up to l^* without interruption, where l^* determines subsequence's ρ

3.4.1.3.1 Proof

$$1, 2, \ldots, u, v, u+1, \ldots, l^* \qquad \text{(Subsequence } S\text{)}.$$

Lemma claims this is suboptimal. Assume it is optimal (proof by contradiction)

Consider two alternate subsequences:

$$v, 1, 2, \ldots, u, u+1, \ldots, l^* \qquad \text{(Subsequence } S'\text{)}.$$

$$1, 2, \ldots, u, u+1, \ldots, l^*, v \qquad \text{(Subsequence } S''\text{)}.$$

Lemma claims both of these are better

$$S \text{ is preferable to } S' \iff \frac{w_1 + \cdots + w_u}{p_1 + \cdots + p_u} > \frac{w_v}{p_v} \qquad (*)$$

$$S \text{ is preferable to } S'' \iff \frac{w_v}{p_v} > \frac{w_{u+1} + \cdots + w_{l^*}}{p_{u+1} + \cdots + p_{l^*}} \qquad (**)$$

Note that since l^* determines $\rho(1, \ldots, k)$,

$$\frac{w_1 + \cdots + w_{l^*}}{p_1 + \cdots + p_{l^*}} \geq \frac{w_1 + \cdots + w_u}{p_1 + \cdots + p_u} \iff$$

$$[(w_1 + \cdots + w_u) + (w_{u+1} + \cdots + w_{l^*})](p_1 + \cdots + p_u)$$
$$\geq (w_1 + \cdots + w_u)[(p_1 + \cdots + p_u) + (p_{u+1} + \cdots + p_{l^*})]$$
$$\iff \frac{w_{u+1} + \cdots + w_{l^*}}{p_{u+1} + \cdots + p_{l^*}} \geq \frac{w_1 + \cdots + w_u}{p_1 + \cdots + p_u} \qquad (***)$$

Suppose S is preferable to S'', then

$$\begin{aligned} \frac{w_v}{p_v} &> \frac{w_{u+1} + \cdots + w_{l^*}}{p_{u+1} + \cdots + p_{l^*}} && \text{from } (**) \\ &\geq \frac{w_1 + \cdots + w_u}{p_1 + \cdots + p_u}. && \text{from } (***) \end{aligned}$$

From $(*)$ this implies that S' is preferable to S, therefore S cannot be optimal. (Note that similarly, we could also show that if S is preferable to S', then S'' is preferable to S.)

This contradicts with the assumption that S is a subsequence of an optimal schedule. Therefore, concludes the proof.

3.4.1.4 Algorithm

If machine free:

1. Start processing chain with highest ρ
2. Process until job l^* (which deterimines ρ) is complete
3. Repeat

3.5 Incurred Costs (Function $h_j(C)$)

3.5.1 $1|prec|h_{\max}$

3.5.1.1 Algorithm

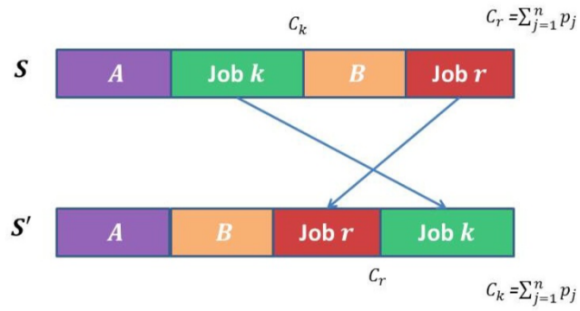
Where J: jobset of jobs that are unscheduled

1. Of unscheduled jobs, create subset J' of jobs that do no need to precede any other unscheduled jobs.

2. Retrieve job k where $h_k\left(\sum_{j\in J} p_j\right) = \min_{l\in J'} h_l\left(\sum_{l\in J} p_j\right)$
3. Schedule job k in the last available slot, where k
4. Recompute J
5. Repeat until J is empty (as in, until all jobs are scheduled)

3.5.1.2 (Brief) Proof Sketch of Scheduling k in last available slot

$$h_k\left(\sum_{j\in J} p_j\right) = \min_{l\in J'} h_l\left(\sum_{l\in J} p_j\right)$$



Just use adjacent pairwise interchange argument to prove that k needs to prove that in the situation above, k should be scheduled last.

3.5.2 Dynamic Programming for $1||\sum_j h_j(C_j)$

3.5.2.1 Time Complexity Savings

$O(n!) \to O(2^n)$

3.5.2.2 Why it Works

Any subsequence is optimal within itself (as in, switching jobs around doesn't make sum of incurred costs any lower)

3.5.2.3 Example

Example $1||\sum_j h_j(C_j)$**:**

j	1	2	3
p_j	4	3	6
$h_j(C_j)$	$C_1 + C_1^2$	$3 + C_2^3$	$8C_3$

To find $V(\{1,2,3\})$, we know:

$$\begin{aligned}
V(\{1\}) &= 4 + 4^2 = 20, \quad V(\{2\}) = 3 + 3^3 = 30, \quad V(\{3\}) = 3(6) = 48 \\
V(\{1,2\}) &= \min(V(\{1\}) + h_2(p_1 + p_2)), V(\{2\}) + h_1(p_1 + p_2)) \\
&= \min(20 + 3 + 7^3, \underline{30 + 7 + 7^2}) = 86, \\
V(\{1,3\}) &= \min(V(\{1\}) + h_3(p_1 + p_3)), V(\{3\}) + h_1(p_1 + p_3)) \\
&= \min(\underline{20 + 8(10)}, 48 + 10 + 10^2) = 100, \\
V(\{2,3\}) &= \min(V(\{2\}) + h_3(p_2 + p_3)), V(\{3\}) + h_2(p_2 + p_3)) \\
&= \min(\underline{30 + 8(9)}, 48 + 3 + 9^3) = 102, \\
V(\{1,2,3\}) &= \min(V(\{1,2\}) + h_3(13)), \\
&\quad \ldots V(\{1,3\}) + h_2(13), V(\{2,3\}) + h_1(13)) \\
&= \min(86 + 8(13), 100 + 3 + 13^3, 102 + 13 + 13^2) = 190.
\end{aligned}$$

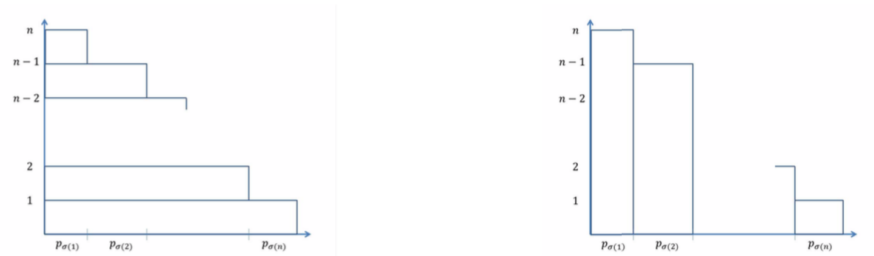
3.6 Misc

3.6.1 Relationship between average completion times and inventory costs

(slide 10/54)

Inventory costs $\propto$ Num jobs in system.

This implies that minimizing avg completion times $\iff$ minimizing in-process inventory levels



$$\begin{aligned}
\text{Area} &= \underbrace{p_{\sigma(1)} + (p_{\sigma(1)} + p_{\sigma(2)}) + \cdots + (p_{\sigma(1)} + \cdots + p_{\sigma(n)})}_{\text{Total completion time}} \\
&= \underbrace{np_{\sigma(1)} + (n-1)p_{\sigma(2)} + \cdots + p_{\sigma(n)}}_{\text{Average number of jobs in system} \times (p_1 + \cdots + p_n)}
\end{aligned}$$

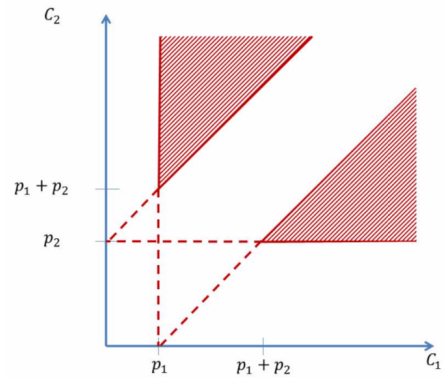
3.7 Modelling Strategies

3.7.1 Disjunctive Constraints (XOR)

3.7.1.1 Description

XOR constraints. They result in a disconnected feasible region (XOR).

3.7.1.1.1 E.g. of disconnected feasible region shown below.



3.7.1.2 Modelling Strategy: Big M

- Choose M such that the constraint it is activated on is essentially nullified
- Use a binary variable as a step function to on/off the addition of M

3.7.1.2.1 Example Problem

$$C_j \geq C_k + p_j \text{ OR } C_k \geq C_j + p_k$$

Essentially, the constraint says that either j_j starts after j_k or the other way around.

3.7.1.2.2 Example Big M Usage We somehow set M to be the completion time of the last job or something like that.

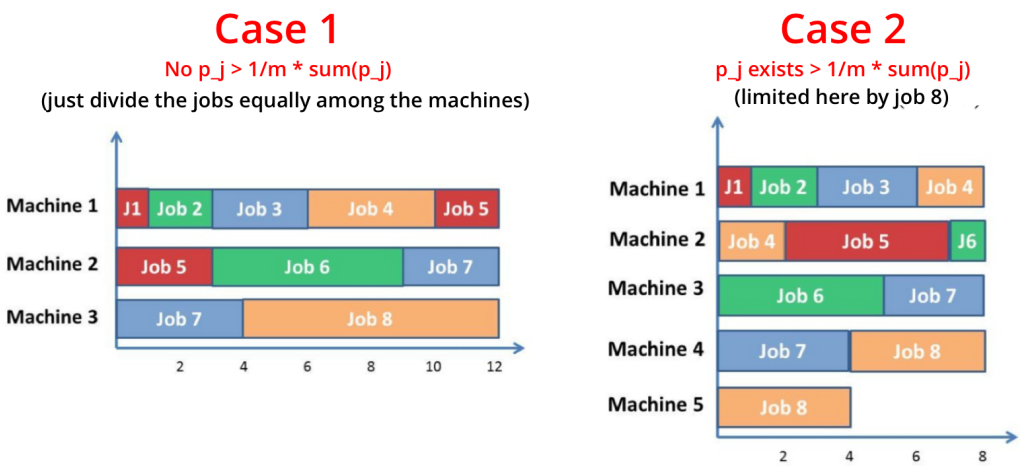
$$MX_{jk} + C_j \geq C_k + p_j$$
$$M(1 - X_{jk}) + C_k \geq C_j + p_k$$

4 L3: $C_{\max}$ for Parallel Machine Environments

4.1 $C_{\max}$ lower bounds

$$C^*_{\max} = \max \left(\underbrace{\max_{j=1,\ldots,n} p_j}_{\text{longest job}} , \underbrace{\frac{\sum_{j=1}^{n} p_j}{m}}_{\substack{\text{jobs allocated evenly} \\ \text{across machines}}} \right)$$

This is obvious when we consider the exhaustive preemptive cases below. Non-preemptive optimal will obviously be $\geq$ the preemptive optimal, so the lower bound is still valid.



4.2 $P_m|prmp|C_{\max}$

4.2.1 Integer Program Formulation

Define X_{ij} := Amount of time machine i spends on job j

$$\begin{aligned}
\min \quad & C_{\max} \\
\text{s.t.} \quad & \sum_{i=1}^{m} X_{ij} = p_j, && \forall j = 1, \ldots, n \\
& \text{(Job } j \text{ to be completed across all machines.)} \\
& C_{\max} \geq p_j, && \forall j = 1, \ldots, n \\
& \text{(Makespan} \geq \text{amount of time each job is processed. Expl. below)} \\
& C_{\max} \geq \sum_{j=1}^{n} X_{ij}, && \forall i = 1, \ldots, m \\
& \text{(Makespan} \geq \text{time each machine processes jobs.} \\
& X_{ij} \geq 0 && \forall i = 1, \ldots, m, \quad \forall j = 1, \ldots, n
\end{aligned}$$

Note: The second constraint seems counter-intuitive at first. It's actually to cover yourself for a Case 2 job (see $C_{\max}$ optimality lower bounds). In this case, without this constraint you risk parallel processing (as in processing at the same time) your longest job.

Note 2: Why we don't care about ensuring no overlap for all non-Case 2 jobs: If we ever have a case of overlap where we can't adjacent swap our way to a feasible solution, then that overlap is indicative of a Case 2 job.

4.2.2 Algorithm

1. Make a non-delay schedule of all jobs on a single machine (in any order)
2. Split this schedule into m parts: $[0, C^*_{\max}], [C^*_{\max}, 2C^*_{\max}], \ldots$ (note that the last part may be shorter)
3. Use interval 1 on machine 1, interval 2 on machine 2, etc etc

Some caveats:

- The last segment may be shorter. This is a limitation of the formulation being an integer formulation
  - e.g. if you have 2 machines and 3 jobs of length 3, you can't have a 4.5/4.5 split between the machines. One has to get 5, one has to get 4
- In case 2 of the lower bound analysis, the intervals are not actually $C^*_{\max}$ long (except for the one containing the overflow of the longest job)

4.3 $P_m||C_{\max}$ (no preemption)

4.3.1 Partition Problem

4.3.1.1 Informal explanation

Can you split a set of positive integers in two such that the sums of the two parts are equal?

4.3.1.2 Formal

- Given:
  - a set of *positive* integers $a_1, a_2, \ldots, a_t$
  - $b = \frac{1}{2}\sum_{j=1}^{t} a_j$
- Does there exist two disjoint subsets of $\{\mathrm{a}_1, \mathrm{a}_2, \ldots, a_\mathrm{t}\}$ such that $\sum_{j_1} a_j = \sum_{j_2} a_j = b$?

4.3.1.3 Relation to $P_2||C_{\max}$

$\gamma(S^*) \leq \frac{1}{2}\sum_{j=1}^{n} p_j \iff$ a solution to the partition problem exists

Note the $\leq$ relaxation of $=$. This is to make it easier for solvers.

4.3.2 Integer Program Formulation

define

$$X_{ij} = \begin{cases} 1, & \text{if machine } i \text{ processes job } j, \quad \forall i = 1, 2, \forall j = 1, \ldots, m \\ 0, & \text{otherwise} \end{cases}$$

$$\begin{array}{lll} \min & C_{\max} & \\ \text{s.t.} & \sum_{j=1}^{n} p_j X_{ij} \leq C_{\max}, & \forall i = 1, \ldots, m \\ & \text{(Makespan is minimally the sum of processing times on machine i)} & \\ & \sum_{j=1}^{m} X_{ij} = 1, & \forall j = 1, \ldots, n \\ & \text{(Each job is processed by exactly 1 machine)} & \\ & X_{ij} \in \{0, 1\} & \forall i = 1, \ldots, m \quad \forall j = 1, \ldots, n \end{array}$$

4.3.3 List Scheduling Algorithm (Heuristic)

4.3.3.1 Algorithm

1. Place jobs in a list (arbitrary ordering)
2. Whenever a machine is idle, unshift the first available job from the list and schedule it to that machine

4.3.3.2 ρ-approximation

$(2 - \frac{1}{m})$-approximation algorithm. Note the obvious optimality when m=1.

4.3.3.3 Proof of ρ-approximation

- Let k be last job to finish in list scheduling algo
- Let $\hat{t}_k$ be the time at which k started processing
- We thus know that at $\hat{t}_k$, all machines must be busy (if there'd been a free machine, it would have taken job k)
- Hence total processing time of all other jobs $\geq m\hat{t}_k$

$$\hat{t}_k \leq \left(\frac{\sum_{j=1}^{n} p_j - p_k}{m} \right) \leq C^*_{max} - \frac{p_k}{m}$$

Also

$$\begin{aligned} C_{max}(LS) &= \hat{t}_k + p_k \leq C^*_{max} - \frac{p_k}{m} + p_k \\ &= C^*_{max} + p_k\,(1 - 1/m) \\ &\underbrace{\leq}_{(lowerbound)} C^*_{max} + C^*_{max}\,(1 - 1/m)\,. \end{aligned}$$

Therefore, we have

$$\frac{C_{max}(LS)}{C^*_{max}} \leq 2 - 1/m.$$

5 L4: Flow Shop Scheduling

5.1 Terminology

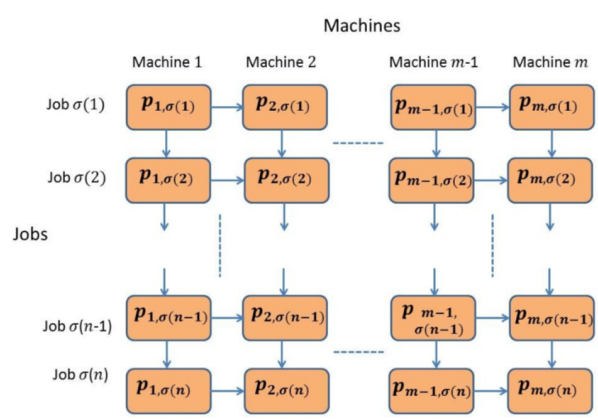
Term	Meaning
Intermediate storage	Buffer between machines where jobs can be kept after finishing on one machine and before starting on the next machine
Blocking flow shop	No intermediate storage (a job not moving to the next machine clogs up the machine it is currently at, even if it is fully done being processed)
Waiting Time	Waiting time of a job in between machines
Idle Time	Waiting time, but for a machine. (More formally, I_{ik} = idle time on a machine between jobs in the rank k and $k+1$

5.2 Misc

- Focus is on makespan objective
- Recall: p_{ij} refers to processing of job j on machine i
- Without being given a permutation, the problem is NP-hard (combinatorial time).
  - e.g. determining processing order of 10 books has 10! possibilities (3,628,800 possibilities)

5.3 Permutation Problem $F_m|prmu|C_{\max}$

5.3.1 Graphical Interpretation



Find the longest path from node $1, \sigma(1)$ to $m, \sigma(n)$, where $p_{i,\sigma(j)}$ is the weight of node $(i, \sigma(j))$ corresponding to processing time.

Personally, I haven't found this very useful. But it makes the exploitable knowledge below easier to see, if you can't already.

5.3.2 Exploitable Knowledge

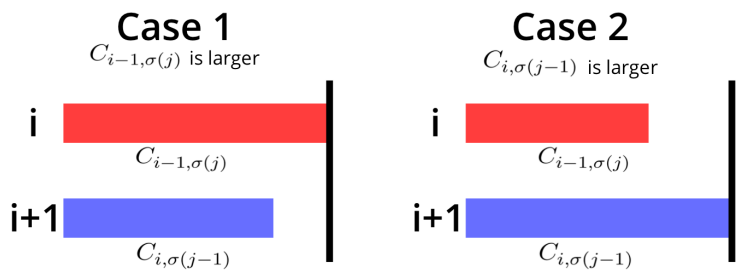
- We know the completion times of job $\sigma 1$ on EACH machine (it's just $\sum_i p_{i,1}$)
- We know the completion times of each job on machine 1 (it's just $\sum_{\sigma(j)} p_{1,j}$)

5.3.3 Recursive Solution

$$C_{i,\sigma(1)} = \sum_{l=1}^{i} p_{l,\sigma(1)}$$
$$C_{1,\sigma(j)} = \sum_{l=1}^{j} p_{1,\sigma(l)}$$
$$C_{i,\sigma(j)} = \max\left(C_{i-1,\sigma(j)}, C_{i,\sigma(j-1)}\right) + p_{i,\sigma(j)}$$

5.3.3.1 Explanation of recursive formulation

Only two cases exist. $\sigma(j)$ can only be processed after the max in each case.



5.4 No Permutation $F||C_{\max}$

5.4.1 Integer Programming

5.4.1.1 Strategy Comments

We introduce two auxilliary variables to help us with the formualation

$I_{i,k}$]: *Idle time* on machine i between jobs in rank k and $k+1$

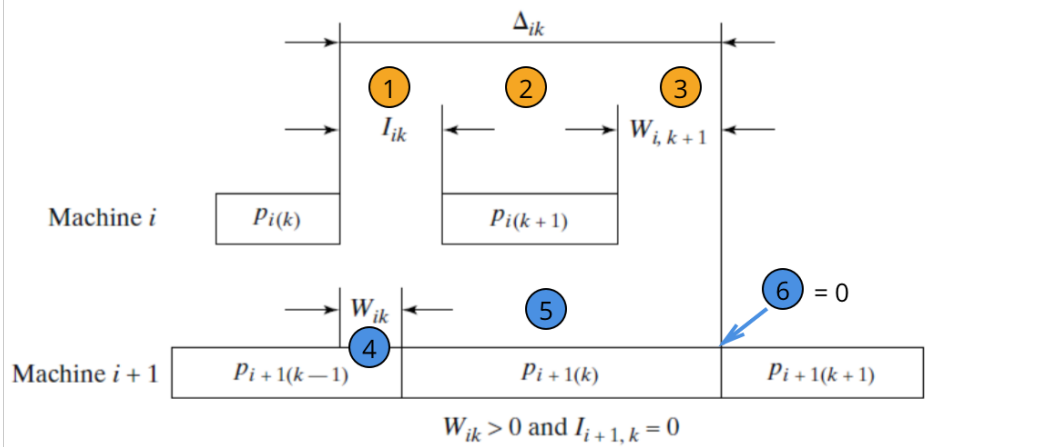
$W_{i,k}$: *Waiting time* on job in rank k between machines i and $i+1$

5.4.1.2 Relationship between $I_{i,k}$ and $W_{i,k}$

The basic idea is to capture the region from $[W_{i,k}, W_{i,k+1}]$.

As in, start of $W_{i,k}$ to end of $W_{i,k+1}$.

$$\overset{1}{I_{ik}} + \overset{2}{\sum_{j=1}^{n} p_{ij} X_{j,k+1}} + \overset{3}{W_{i,k+1}} \quad = \quad \overset{4}{W_{ik}} + \overset{5}{\sum_{j=1}^{n} p_{i+1,j} X_{jk}} + \overset{6}{I_{i+1,k}},$$
$$\forall i = 1, \ldots, m-1, \forall k = 1, \ldots, n-1.$$



Note that either $W_{i,k+1} = 0$ Or $I_{i+1,k} = 0$. Or both $= 0$, but that's unlikely.

There are 2 possible cases. Technically, the third both=0 case too, but let's leave that aside. It's easy to understand after the first 2 cases.

These are the cases from the recursion explanation. Essentially, the effect is that:

$W_{i,k+1} > 0 \implies I_{i+1,k} = 0$ as, if job $k+1$ has to wait to enter machine $i+1$ as soon as it can, then there's no pause between machine $i+1$ processing jobs k and $k+1$.

$I_{i+1,k} > 0 \implies W_{i,k,k+1} = 0$ as, if machine $i+1$ has to wait before processing job $k+1$ as soon as it can, this means that job $k+1$ is still being processed on machine i when machine $i+1$ is free, and thus there is no pause between it being processed on machines i and $i+1$

5.4.1.3 Idea for alternate formulation of objective function

Selin's original formulation is from the perspective of the last machine.

My alternate one (already checked with her, it's correct) is from the perspective of the last job.

This also hints at a symmetry present: you can swap the jobs and the machines and effecively, you still have the same optimal value. Selin said this never makes solving it any faster, but I have a feeling she might be wrong (and she didn't seem super confident while she said it, anyway)

5.4.1.4 Naive Objective Function

$$\min \quad \underbrace{\sum_{i=1}^{m-1} \sum_{j=1}^{n} p_{ij} X_{j1}}_{\text{Start time on 1st job on machine m}} + \underbrace{\sum_{j=1}^{n-1} I_{mj}}_{\text{Idle time of machine m}} + \underbrace{\sum_{j=1}^{n} p_{mj}}_{\text{Processing time of all jobs on machine m}}$$

However, the processing times on the last machine (the final term) is constant so we can effectively just remove it

5.4.1.5 Full Formulation

$$
\begin{aligned}
\min \quad & \sum_{i=1}^{m-1} \sum_{j=1}^{n} p_{ij} X_{j1} + \sum_{k=1}^{n-1} I_{mk} \\
\text{s.t.} \quad & \sum_{j=1}^{n} X_{jk} = 1 && \forall k = 1, \ldots, n \\
& \text{(Each position has exactly one job)} \\
& \sum_{k=1}^{k} X_{jk} = 1 && \forall j = 1, \ldots, n \\
& \text{(Each job has exactly one position)} \\
& I_{i,k} + \sum_{j=1}^{n} p_{i,j} X_{j,k+1} + W_{i,k+1} = W_{i,k} + \sum_{j=1}^{n} p_{i+1,j} X_{j,k} + I_{i+1,k} && \forall i = 1, \ldots, m-1, \quad \forall k = 1, \ldots, n-1 \\
& \text{(Define W and I through shared relationships with } p_{ij}\text{)} \\
& I_{1,k} = 0 && \forall \forall k = 1, \ldots, n-1 \\
& \text{(No idle time on first machine (on all jobs!)} \\
& W_{i,1} = 0 && \forall i = 1, \ldots, m-1 \\
& \text{(No waiting time for first job (on all machines!)} \\
& X_{j,k} \in \{0, 1\} && \forall j = 1, \ldots, n, \quad \forall k = 1, \ldots, n \\
& I_{i,k} \geq 0 && \forall i = 1, \ldots, m, \quad \forall k = 1, \ldots, n-1 \\
& \text{(k stops at } n-1 \text{ as job } n+1 \text{ does not exist)} \\
& W_{i,k} \geq 0 && \forall i = 1, \ldots, m-1, \quad \forall k = 1, \ldots, n \\
& \text{(i stops at } m-1 \text{ as machine } m+1 \text{ does not exist)}
\end{aligned}
$$

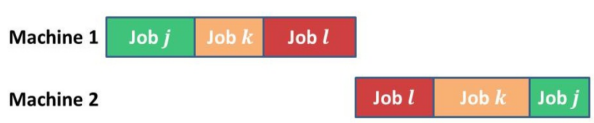
5.4.2 Theorem: Sufficient to consider schedules which maintain processing order (2-3 machines)

Also works for the first/last 2 machines.

5.4.2.1 Intuition and Proof Sketch

Basically, in the 2-machine case, if you swap the job order between $\sigma(k)$ and $\sigma(k+1)$ on the second machine only, then the entire C_max gets delayed waiting for $p_{1,\sigma(k+1)}$.

TLDR: You introduce unforced idleness, and that's retarded.



5.4.2.2 Why this works for 3 machines

We know it works for 2 machines, from the proof.

- See 3 machines as having 2 sets of 2 machines with overlap (first 2 machines, last 2 machines).
  - By the proof, first 2 have same order and last two have same order
  - As the middle job is shared, that means all 3 machines have the same order

If this is true, though, then why doesn't it work for 4 machines (3 on each end), and so on for 5 and so forth till all machines maintain the same order?

- Answer: Because the 3-machine logic breaks if you have 4 jobs. 3-machine logic only works because the 2-machine pairs at the end have an intersection (the middle machine). If we have 4 machines, that intersection is gone.

5.4.3 Idle Time Theorem for 2 machines

Where X_j is the idle time of the 2nd machine immediately before the jth job in the sequence is processed. Obviously, machine 1 has no idle time.

$$I_n = \sum_{j=1}^{n} X_j = \max\left(\max_{t=1,\ldots,n}\left(\sum_{j=1}^{t} p_{1j} - \sum_{j=1}^{t-1} p_{2j}\right), 0\right)$$

The total makespan $C_{\max} = I_n + \sum_{j=1}^{n} p_{2j}$

5.4.3.1 Corollary 1

Minimizing $C_{\max}$ is eq.. to minimizing idle time on machine 2 (which is the same as minimizing X_n)

5.4.3.2 Corollary 2

Decreasing all processing times by a common constant does not change the optimal sequence

5.4.4 Johnson's Rule

5.4.4.1 Algorithm

1. List jobs and processing times at each of the two machines
2. Select job with shortest activity time among the $2n$ processing times. a. If that p is for machine 1, schedule that job first b. Else, schedule that job last
3. Delete job from list
4. Repeat

5.4.4.2 Notes

At iteration i, there are $2(n-i)$ processing times left. You're working from the ends to the middle

5.4.4.3 Example

Example $\mathbf{F}_2||C_{\max}$

j	1	2	3	4	5
p_{1j}	4	4	30	6	2
p_{2j}	5	1	4	30	3

1. Smallest time is job 2 on machine 2.

				2

2. Eliminate job 2, next smallest is p_{51}.

5				2

3. Eliminate job 5, next smallest is p_{11}.

5	1			2

4. Eliminate job 1, next smallest is p_{32}.

5	1		3	2

5. Eliminate job 3, place job 4 as third.

5	1	4	3	2

5.4.4.4 Proof

5.4.4.4.1 Comments

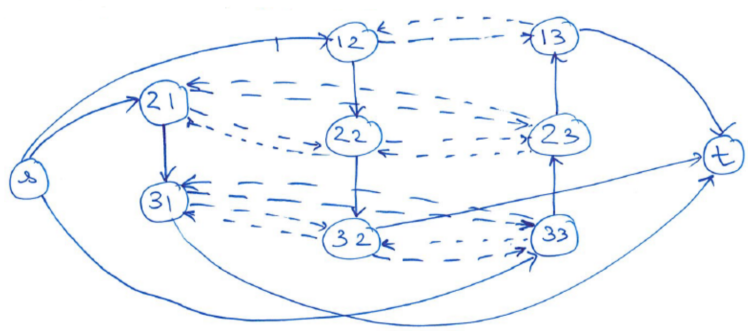
- I don't think this proof actually does its job
  - Problem in its demo of JR holding for $n = 1$ case. You could easily invert Johnson's rule and it would still be true.

6 TODO L5: Job Shop Scheduling

6.1 Disjunctive Graph Formulation

Job	p_{j1}	p_{j2}	p_{j3}	Processing Order
1	5	5	3	M2 → M3
2	4	4	2	M1 → M2 → M3
3	4	4	3	M3 → M2 → M1

Note the irrelevance of p_{11}. Idk why it's there, probably a mistake in the slides.



6.2 TODO Integer Program Formulation (Based on Disjunctive Graph)

This is pretty tough to generalize and write concisely. Here's Selin's generalization, but it's not super clear.

7 TODO L6: Uncertainty in Scheduling (Stochastic)

7.1 General

- We minimize expected costs
- Generally, we know (or assume) a particular distribution in order to do the math.
- Usually, for random variables we put a tilde on top (e.g. $\tilde{p}_j$ is stochastic, while p_j is not)

7.2 Potential Areas of Randomness

- p_j
- r_j
- possibilities of machines breaking down

8 Proving Techniques

8.1 Induction

1. Prove base case (e.g. show P(0) holds)
2. Prove inductive step for general case (e.g. show if P(n) holds then P(n+1) holds)

8.2 Enumeration/Construction/Direct (brute force)

Using definition, show it's always true

8.3 Contradiction (very useful!)

1. Assume counter claim (e.g. theorem says $P \implies Q$, then assume $P \implies \neg Q$)
2. Show contradiction in assumption

Particularly useful is the **adjacent pairwise interchange argument** (swap abstract adjacent jobs to demonstrate your point mathematically)

8.4 Contra-positive

Show $A \implies B$ due to $\neg B \implies \neg A$

9 TODO To Revise

9.1 Newsvendor

9.2 Dynamic Programming

9.2.1 Components

- State
- Stage (optional)
- Value function (forward looking means look at future state, backward looking means look at backward state)
- Forward or backward
- Transfer function (governs state to state transition as a function of actions)

9.2.2 Example

min $1||\sum_j h_j(C_j)$

State: Set of all jobs already scheduled $:= S$

Value function: $V(S) = \min_k h_k(\sum_{J \in S} p_J) + V(S \setminus k)$

9.2.3 DP-tacklable problems

They have an optimality property that allows you to optimize subproblems in order to construct the overall optimal solution

Obviously, this is not a rigorous definition,

9.3 TODO Simplex

9.4 TODO Dijkstra

9.5 TODO Branch and Bound

10 Misc

10.0.1 Story

Paperboy wants to sell papers. Needs to balance opportunity cost and cost of unsold newspapers.

The goods are **perishable** (expires after today).

10.0.2 Formulation

$$\min\{c^w \max(0, p - x) + c^I \max(0, x - p)\}$$

10.0.3 Optimal Solution

$$x = F^{-1}\left(\frac{c^W}{c^W + c^I}\right)$$

Note that F is the CDF.

Basically, we differentiated and set to 0. It's analytically evident that the formulation is convex.

11 AMPL

11.1 General

- A Mathematical Programming Language (AMPL)
- Allows for decoupling of the following elements:
  - mathematical programming model
  - data
  - solver

11.2 Tips

- `option omit_zero_rows 1;`
- `option relax_integrality 1;`
- `reset data;`
- `update data;`