Project Planning & Scheduling

ISE 419

1

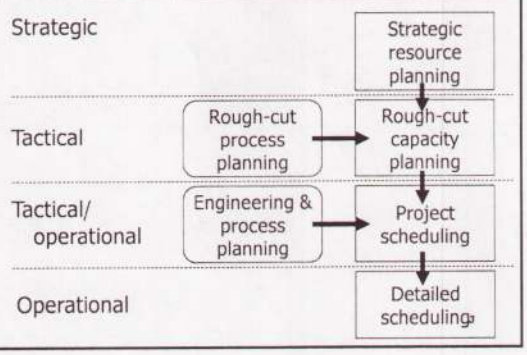
Project Scheduling

- Project definition:

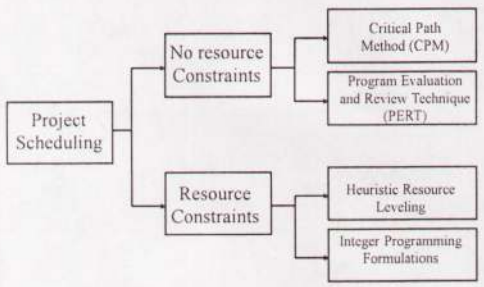
A complex and large scale one-of-a-kind product or service, made up by a number of component activities (jobs), that entails a considerable financial effort and must be time-phased, i.e. scheduled, according to specified precedence and resource requirements

2

Hierarchical planning



Overview



4

CPM and PERT

Network models can be used as an aid in the scheduling of large complex projects that consist of many activities.

CPM: If the duration of each activity is known with certainty, the Critical Path Method (CPM) can be used to determine the length of time required to complete a project.

PERT: If the duration of activities is not known with certainty, the Program Evaluation and Review Technique (PERT) can be used to estimate the probability that the project will be completed by a given deadline.

5

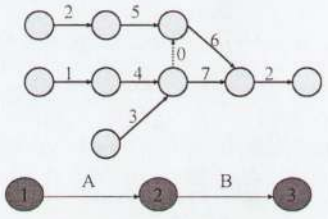
CPM and PERT are used in many applications including the following:

- Scheduling construction projects such as office buildings, highways and swimming pools
- Developing countdown and "hold" procedure for the launching of space crafts
- Installing new computer systems
- Designing and marketing new products
- Completing corporate mergers
- Building ships

6

Project Scheduling

- Jobs subject to precedence constraints
- Job on arc format (most common)



Activity A must be completed before activity B starts

7

While constructing an AOA type of project diagram one should use the following rules:

1. Node 1 represents the start of the project. An arc should lead from node 1 to represent each activity that has no predecessors.
2. A node (called the finish node) representing the completion of the project should be included in the network.
3. Number the nodes in the network so that the node representing the completion time of an activity always has a larger number than the node representing the beginning of an activity.
4. An activity should not be represented by more than one arc in the network
5. Two nodes can be connected by at most one arc. To avoid violating rules 4 and 5, it can be sometimes necessary to utilize a *dummy activity* that takes 0 time.

8

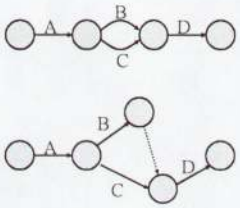
Planning a Concert

Task		Predecessors
A	Plan concert	-
B	Advertise	A
C	Sell tickets	A
D	Hold concert	B, C

9

Job on Arc Network

- *Not allowed*: no two jobs can have the same starting and ending node!
- Need to introduce a dummy job.

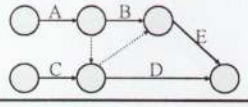


Task		Predecessors
A	Plan concert	-
B	Advertise	A
C	Sell tickets	A
D	Hold concert	B, C

10

Changing a Tire

Task		Predecessors
A	Remove flat tire from wheel	-
B	Repair puncture on flat tire	A
C	Remove spare from trunk	-
D	Put spare on wheel	A, C
E	Place repaired tire in trunk	B, C



11

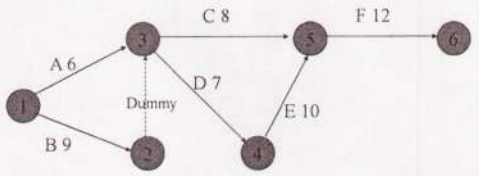
An example for CPM

Widgetco is about to introduce a new product. A list of activities and the precedence relationships are given in the table below. Draw a project diagram for this project.

Activity	Predecessors	Duration(days)
A:train workers	-	6
B:purchase raw materials	-	9
C:produce product 1	A, B	8
D:produce product 2	A, B	7
E:test product 2	D	10
F:assemble products 1&2	C, E	12

12

Project Diagram for Widgetco

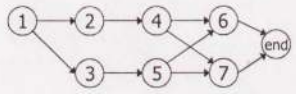


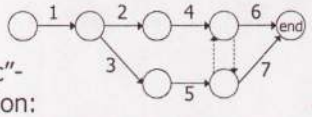
Node 1 = starting node
Node 6 = finish node

Project representation example

Job	Duration	Predecessors
1	2	-
2	3	1
3	1	1
4	4	2
5	2	3
6	1	4,5
7	3	4,5

"job on node"-representation:



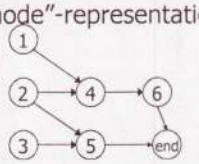


"job on arc"-representation:

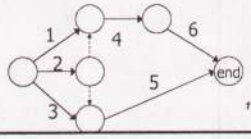
Project representation example

Job	p(j)	Predecessors
1	2	-
2	3	-
3	1	-
4	4	1,2
5	2	2,3
6	1	4

"job on node"-representation:



"job on arc"-representation:



Critical Path Method

- An analytical tool that provides a schedule that completes the project in minimum time subject to the precedence constraints. In addition, CPM provides:
  - Starting / ending times for each activity
  - Identification of the critical activities (i.e., the ones whose delay necessarily delay the project).
  - Identification of the non-critical activities, and the amount of slack time available when scheduling these activities.
- Think of unlimited machines in parallel
  - … and n jobs with precedence constraints
- Objective to minimize makespan

Critical Path Method (CPM)

- Critical job: job without slack

S_j = earliest possible starting time of job j
S_j^* = latest possible starting time of job j
C_j = earliest possible completion time of job j
C_j^* = latest possible completion time of job j
$\text{slack}_j = C_j^* - p_j - S_j$

- Critical path = chain of critical jobs

Critical Path Method (CPM)

- Critical path method initialization:
  - determine earliest starting time for all jobs
  - determine latest completion time for all jobs
  - determine which jobs have no slack
- 2 CPM solution methods:
  - Forward procedure
  - Backward procedure

Critical Path Method

- Forward procedure:
  - Starting at time zero, calculate the **earliest** each job can be started
  - The completion time of the last job is the makespan
- Backward procedure
  - Starting at time equal to the makespan, calculate the **latest** each job can be started so that this makespan is realized

Forward Procedure

Step 1:

Set at time $t = 0$ for all jobs j with no predecessors,

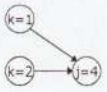
$S_j^{'}=0$ and set $C_j^{'} = p_j$.

Step 2:

Compute for each job j

$$S_j^{'} = \max_{\text{all } k \to j} C_k^{'},$$

$$C_j^{'} = S_j^{'} + p_j.$$



Step 3:

The optimal makespan is $C_{\max} = \max\{C_1^{'}, C_2^{'}, ..., C_n^{'}\}$

STOP

Backward Procedure

Step 1:

Set at time $t = C_{max}$ for all jobs j with no successors,

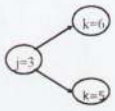
$C_j^{''} = C_{max}$ and set $S_j^{''} = C_{max} - p_j$.

Step 2:

Compute for each job j

$$C_j^{''} = \min_{j \to \text{all } k} S_k^{''},$$

$$S_j^{''} = C_j^{''} - p_j.$$



Step 3:

Verify that

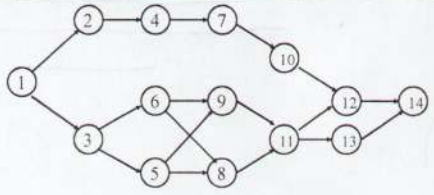
STOP $0 = \min\{S_1^{''}, ..., S_n^{''}\}$.

Comments

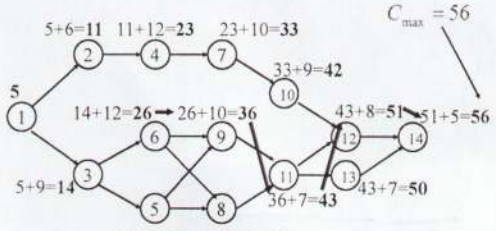
- The forward procedure gives the earliest possible starting time for each job
- The backwards procedures gives the latest possible starting time for each job
- If these are equal the job is a **critical job**.
- If these are different the job is a **slack job**, and the difference is the **float**.
- A **critical path** is a chain of jobs starting at time 0 and ending at C_{max}

Example

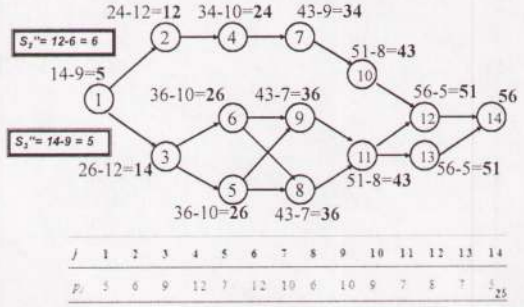
j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5



Forward Procedure



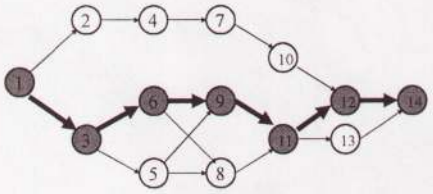
Backwards Procedure



j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5

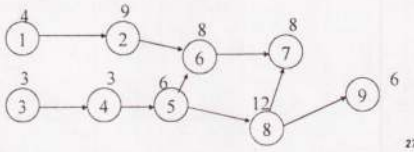
Σ_{25}

Critical Path



Another Example

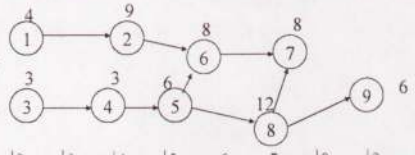
jobs	1	2	3	4	5	6	7	8	9
p_j	4	9	3	3	6	8	8	12	6



Forward and Backward Passes

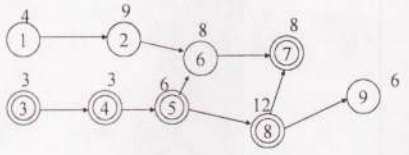
$S_j = \max_{i \in k \to j} C_i$,

$C_j = \min_{j \to k \in k} S_k$,



jobs	1	2	3	4	5	6	7	8	9
S_j	0	4	0	3	6	max {13,12} = 13	max {21,24} =24	12	24
C_j	4	4+9=13	3	3+3=6	6+6=12	13+8= 21	24+8= 32	12+12= 24	24+6= 30
C''_j	7	16	3	6	min {16,12} =12	24	32	min {24,26} =24	32
S''_j	7-4=3	16-9=7	3-3=0	6-3=3	12-6=6	24-8=16	32-8=24	24-12=12	32-6=26

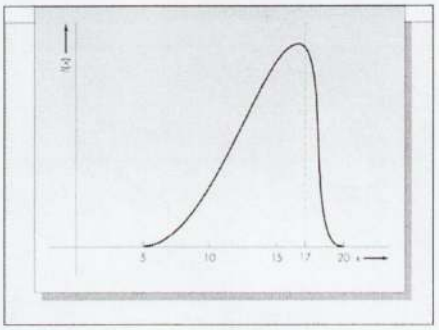
Critical Path



PERT: Project Evaluation and Review Technique

- PERT is a generalization of CPM to allow for uncertain activity times. For each activity the user must specify:
  - a = minimum completion time
  - b = maximum completion time
  - m = most likely completion time
- The method assumes each activity time follows a beta distribution, which can be fit precisely with specification of a, b, and m.
- The following figure provides an example with a= 5, b=20 and m=17).

Probability Density of Activity Time



31

PERT

- The mean and standard deviation of activity times are estimated from the following formulas (based on the beta distribution)

$$\mu = \frac{a+4m+b}{6} \text{ and } \sigma = \frac{b-a}{6}$$

- In PERT one assumes that the path the with longest expected completion time is the true critical path *(this is only an approximation, since true critical path is a random variable).*

32

PERT

- PERT requires the assumption that the durations of all activities are independent. Thus,

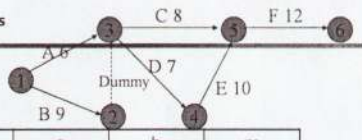
$\sum_{i \in path} E(T_i)$: expected duration of activities on any path

$\sum_{i \in path} \text{var } T_i$: variance of duration of activities on any path

- Finally, one invokes the *Central Limit Theorem* to conclude that the total project completion time is a random variable whose distribution is approximately normal.

33

Example:
a, b and m for activities in Widgetco



Activity	a	b	m
A	2	10	6
B	5	13	9
C	3	13	8
D	1	13	7
E	8	12	10
F	9	15	12

34

According to the table on the previous slide:

$$E(T_A) = \frac{2+10+24}{6} = 6, \text{var } T_A = \frac{(10-2)^2}{36} = 1.78$$

$$E(T_B) = \frac{5+13+36}{6} = 9, \text{var } T_B = \frac{(13-5)^2}{36} = 1.78$$

$$E(T_C) = \frac{3+13+32}{6} = 8, \text{var } T_C = \frac{(13-3)^2}{36} = 2.78$$

$$E(T_D) = \frac{1+13+28}{6} = 7, \text{var } T_D = \frac{(13-1)^2}{36} = 4$$

$$E(T_E) = \frac{8+12+40}{6} = 10, \text{var } T_E = \frac{(12-8)^2}{36} = 0.44$$

$$E(T_F) = \frac{9+15+48}{6} = 12, \text{var } T_F = \frac{(15-9)^2}{36} = 1$$

35

- Of course, the fact that arc (2,3) is a dummy arc yields

$E(T_{23})=\text{var}T_{23}=0$

- The critical path is 1-2-3-4-5-6.
- Thus,

E(CP)=9+0+7+10+12=38
varCP=1.78+0+4+0.44+1=7.22
Then the standard deviation for CP is $(7.22)^{1/2}=2.69$

$$P(CP \le 35) = P\left(\frac{CP-38}{2.69} \le \frac{35-38}{2.69}\right) = P(Z \le -1.12) = 0.13$$

PERT implies that there is a 13% chance that the project will be completed within 35 days.

36

Discussion

- Potential problems with PERT:
  - Always underestimates project duration
    - » other paths may delay the project
  - Non-critical paths ignored
    - » critical path probability
    - » critical activity probability
  - Activities are not always independent
    - » same raw material, weather conditions, etc.
  - Estimates may be inaccurate

37

Improved PERT parameter estimates

Reference: H.-S. Lau and H.-L. Lau, An Improved PERT-type Formula for Standard Deviation. IIE Transactions, 30, 1998, 273-275.

- Improvement over "extended Pearson-Tukey" formula.
- Pearson-Tukey mean:

$$\mu = .63x_{0.5} + .185[x_{0.05} + x_{0.95}]$$

where x_α = the α percentile of R. V. x distribution

- Original variance estimate assumed a Beta dist
  - Special case of Pearson family of distributions
  - "many, if not most, empirical distributions fall outside Beta area..."

38

Improved PERT parameter estimates

- Form of improved variance estimate

$$\sigma = c_1 x_\alpha + c_2 x_{0.5} + c_3 x_{1-\alpha}$$

- Regression used to get parameter estimates
  - 5000 randomly generated Pearson distributions sampled within "most applicable" range (best)
  - Alphas considered: .01, .025, .05, .1, .25
- Positive skew:

$$\sigma = .34[x_{0.95} - x_{0.5}] + .28[x_{0.5} - x_{0.05}]$$

- Negative skew:

$$\sigma = .28[x_{0.95} - x_{0.5}] + .34[x_{0.5} - x_{0.05}]$$

39

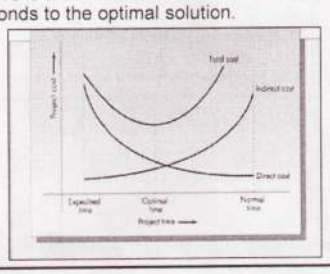
Time Costing Methods

- Suppose that projects can be expedited by reducing the time required for critical activities.
  - Doing so results in an increase in some costs and a decrease in others.
  - The goal is to determine the optimal number of days to schedule the project to minimize total cost.

40

Optimal Project Completion Time

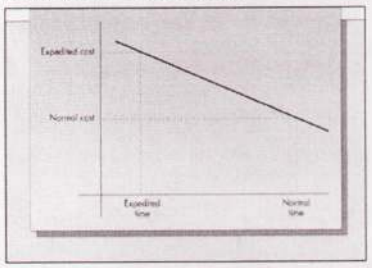
- Since direct costs decline with the project time and indirect costs increase with the project time, the total cost curve is a convex function whose minimum corresponds to the optimal solution.



41

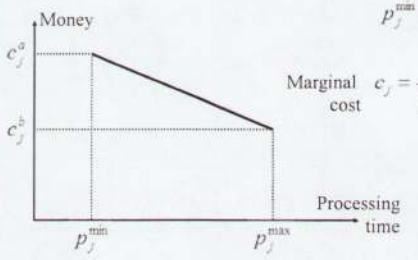
The CPM Cost-Time Linear Model

- Assume that there is a linear time/cost relationship for each activity.



42

Linear Costs



$$p_j^{min} \le p_j \le p_j^{max}$$

Marginal cost $c_j = \dfrac{c_j^a - c_j^b}{p_j^{max} - p_j^{min}}$

43

Problem

- Spend money to reduce processing times so as to minimize:

$$c_0 C_{max} + \sum_{j=1}^{n} \left(c_j^b + c_j \left(p_j^{max} - p_j \right) \right)$$

"Overhead" cost — Cost per activity

44

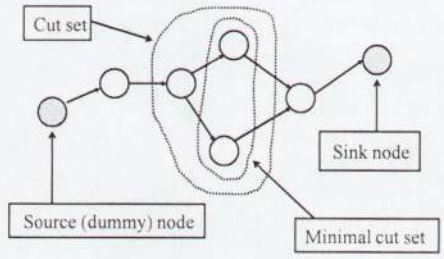
Time/costs trade-off heuristic

Definitions:

- source and sink in precedence graph
- critical path: longest path from source to sink
- G_{cp} = sub-graph of critical path(s)
- cut set: set of nodes in sub-graph G_{cp} whose removal results in disconnecting the source from the sink in the precedence graph
- minimal cut set: if putting back 1 node in the graph connects the source to the sink

45

Sources, Sinks, and Cuts



46

Time/costs trade-off heuristic

STEP 1: Set $p_j = p_j^{max}$ $(\forall j)$
Determine G_{cp} *(Critical Path Method)*

STEP 2: Determine all *minimum cut sets* mcs in G_{cp}.
Consider only those mcs with all processing times $p_j > p_j^{min}$
If there is no such set: STOP

STEP 3: For each mcs compute the costs of reducing all p_j in mcs by 1 time unit.
Let mcs* be the mcs with the lowest costs
If the lowest costs are $< c_0 \Rightarrow$ apply the changes

STEP 4: Revise G_{cp} by reducing proc. times in the mcs by 1 and finding new set of critical paths. Go to STEP 2

47

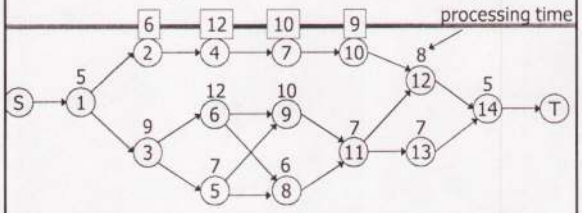
Example

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
p_j^{max}	5	6	9	12	7	12	10	6	10	9	7	8	7	5
p_j^{min}	3	5	7	9	5	9	8	3	7	5	4	5	5	2
c_j^a	20	25	20	15	30	40	35	25	30	20	25	35	20	10
c_j	7	2	4	3	4	3	4	4	4	5	2	2	4	8

c_0 = fixed overhead costs per time unit = 6

48

Time/costs trade-off heuristic example



longest path = $56 = 1 \rightarrow 3 \rightarrow 6 \rightarrow 9 \rightarrow 11 \rightarrow 12 \rightarrow 14$

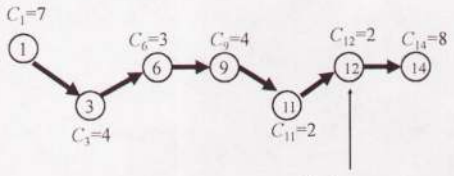
cut sets: $\{1\},\{3\},\{6\},\{9\},\{11\},\{12\},\{14\}$

minimum cut sets with lowest costs: $\{11\}$ and $\{12\}$

$c_{12} = 2 < c_0 = 6 \Rightarrow$ *net savings* of $4 \Rightarrow p_{12} := 7$

49

Critical Path Subgraph (G_{cp})

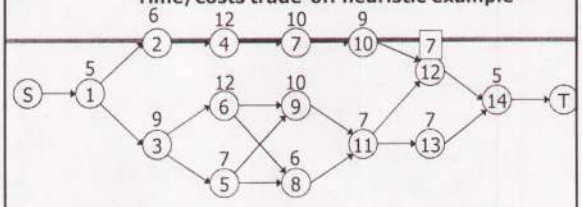


Cut sets: {1},{3},{6},{9}, {11},{12},{14}.

Minimum cut set with *lowest cost*

50

Time/costs trade-off heuristic example



longest path = $55 : 1 \rightarrow 3 \rightarrow 6 \rightarrow 9 \rightarrow 11 \rightarrow \{12,13\} \rightarrow 14$

minimum cut sets: $\{1\},\{3\},\{6\},\{9\},\{11\},\{12,13\},\{14\}$

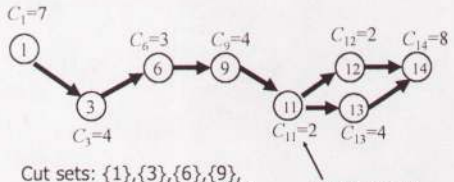
minimum cut set with lowest costs: $\{11\}$

$c_{11} = 2 < c_0 = 6 \Rightarrow$ *net savings* of 4

$\Rightarrow p_{11} := 6$

51

Critical Path Subgraph (G_{cp})

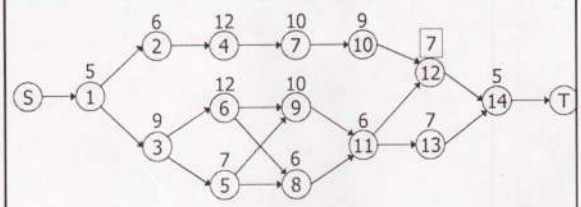


Cut sets: {1},{3},{6},{9}, {11},{12,13},{14}.

Minimum cut set with lowest cost

52

Time/costs trade-off heuristic example

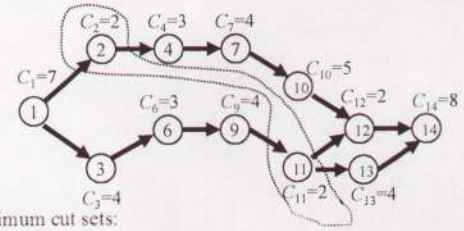


longest path = $54 : 1 \rightarrow 3 \rightarrow 6 \rightarrow 9 \rightarrow 11 \rightarrow \{12,13\} \rightarrow 14$

or : $1 \rightarrow 2 \rightarrow 4 \rightarrow 7 \rightarrow 10 \rightarrow 12 \rightarrow 14$

53

Critical Path Subgraph (G_{cp})



minimum cut sets:

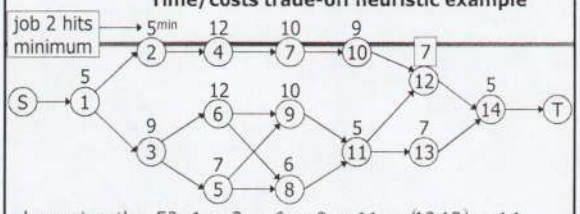
$\{1\},\{i,j\} : i \in \{2,4,7,10,12\} + j \in \{3,6,9,11\},\{12,13\},\{14\}$

minimum cut set with lowest costs: $\{2,11\}$

$c_2 + c_{11} = 4 < c_0 = 6 \Rightarrow$ *net savings* of $2 \Rightarrow p_2 := 5^*, p_{11} := 5$

54

Time/costs trade-off heuristic example



longest path = 53 : $1 \to 3 \to 6 \to 9 \to 11 \to \{12,13\} \to 14$

or : $1 \to 2 \to 4 \to 7 \to 10 \to 12 \to 14$

minimum cut sets:

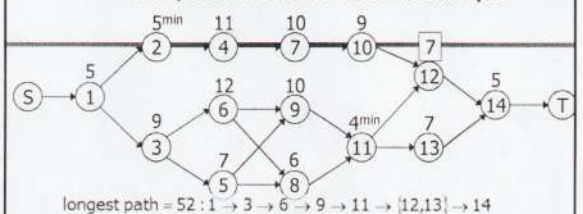
$\{1\}, \{i,j\} : i \in \{4,7,10,12\} + j \in \{3,6,9,11\}, \{12,13\}, \{14\}$

minimum cut set with lowest costs: $\{4,11\}$

$c_4 + c_{11} = 5 < c_0 = 6 \Rightarrow$ *net savings* of $1 \Rightarrow p_4 := 11, p_{11} := 4$ *

55

Time/costs trade-off heuristic example



longest path = 52 : $1 \to 3 \to 6 \to 9 \to 11 \to \{12,13\} \to 14$

or : $1 \to 2 \to 4 \to 7 \to 10 \to 12 \to 14$

minimum cut sets:

$\{1\}, \{i,j\} : i \in \{4,7,10,12\} + j \in \{3,6,9\}, \{12,13\}, \{14\}$

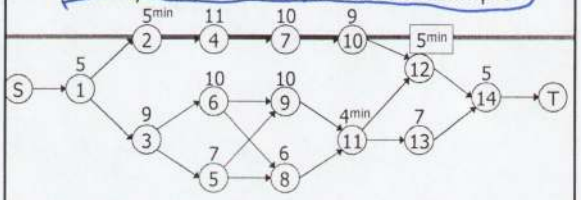
minimum cut set with lowest costs: $\{6,12\}$ ... reduce each by 2

$2(c_6 + c_{12}) = 10 < c_0 = 2(6) = 12$

$\Rightarrow$ net savings = $2 \Rightarrow p_6 := 10, p_{12} = 5(\min)$

56

Time/costs trade-off heuristic example



longest path = $50 : 1 \to 3 \to 6 \to 9 \to 11 \to 13 \to 14$

or : $1 \to 2 \to 4 \to 7 \to 10 \to 12 \to 14$

minimum cut sets:

$\{1\}, \left[\{i,j\} : i \in \{4,7,10\} + j \in \{3,6,9\}\right], \{14\}$

minimum cut set with lowest costs: $\{4,6\}$

$c_4 + c_6 = 6$ *not* $< c_0 = 6 \Rightarrow$ *no net savings* $\Rightarrow$ *STOP*

57

Time/costs trade-off LP model

Decision variables:

x_j = earliest starting time of job j

p_j = processing time of job j

Total cost: $\min c_0 C_{max} + \sum_j \left(c_j^b + c_j (p_j^{max} - p_j) \right)$

Constraints:

precedence relations: $x_k - p_j - x_j \geq 0$ (if $j \to k$)

max/min processing times: $p_j \leq p_j^{max}, p_j \geq p_j^{min}$

C_{max} determination: $C_{max} \geq x_j + p_j$ ($\forall j$)

$x_j \geq 0$

58

Time/costs trade-off: nonlinear costs

Discrete time-framework:

- decreasing convex cost-function
  $\Rightarrow c_j(p_j - 1) - c_j(p_j) \geq c_j(p_j) - c_j(p_j + 1)$
- non-decreasing overhead cost-function $c_0(t)$
- use the same heuristic as for the linear costs

Continuous time-framework:

- non-linear model with the same constraints as the LP-model, but with the objective:

$$\int_0^{Cmax} c_0(t)\,dt + \sum_j c_j(p_j)$$

59

Single and Parallel Machine Scheduling

IE 419

Supplementary Reference:
M. Pinedo, "Scheduling: Theory, Algorithms, and Systems, 3rd Ed.", Springer, 2008
http://users.utu.fi/yurnik/scheduling_files/Scheduling Theory, Algorithms, and Systems_Michael_Pinedo.pdf
http://www.ebook3000.com/Scheduling--Theory--Algorithms--and-Systems_22332.html

Modeling Parameters

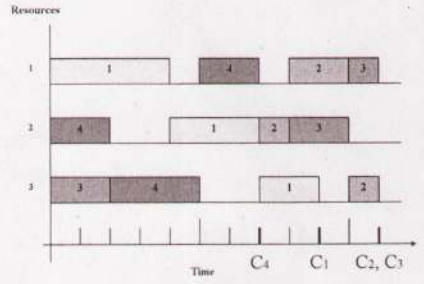
Typical scheduling parameters:

- Number of resources (m machines, operators)
- Configuration and layout
- Resource capabilities
- Number of jobs (n)
- Job processing times (p_{ij})
- Job release and due dates (resp. r_{ij} and d_{ij})
- Job weight (w_{ij}) or priority
- Setup times

Decision Outcomes

- Three basic types of solutions:
  - A **sequence**: a permutation of the jobs
  - A **schedule**: allocation of the jobs in a more complicated setting of the environment
  - A **scheduling policy**: determines the next job given the current state of the system

Emphasis on *Completion time*



Optimality criteria

We define for each job j:

C_{ij} **completion time** of the operation of job j on machine i

C_j time when job j exits the system 系统完成退出系统的时间.

$L_j = C_j - d_j$ **lateness** of job j 如果提高完成则为负.

$T_j = \max(C_j - d_j, 0)$ **tardiness** of job j

$U_j = \begin{cases} 1 & \text{if } C_j > d_j \\ 0 & \text{otherwise} \end{cases}$ unit penalty of job j

Optimality criteria

Possible objective functions to be minimised:

Makespan C_{max} - $\max(C_1,..,C_n)$

Maximum lateness L_{max} - $\max(L_1,..,L_n)$

Total weighted completion time $\sum w_j C_j$ - weighted flow time

Total weighted tardiness $\sum w_j T_j$

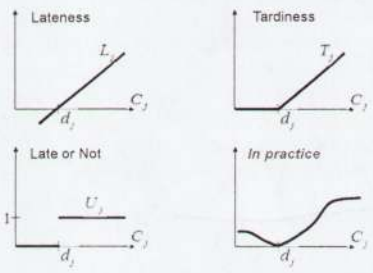
Weighted number of tardy jobs $\sum w_j U_j$

- ***Flowtime***
  - *Time job j spends in the system*
  - Flow time: $F_j = C_j - r_j$

1) Minimize average flowtime
2) Minimize maximum flowtime

真正 j completed.
w_j 才消失.
Thus. $w_j C_j$ ✓
$w_j P_j$ X.

Due Date Penalties



7

Generic notation of scheduling problem

Scheduling problem: α | β | γ	
α	machine environment
β	job characteristics
γ	optimality criteria

for example:

$Pm \mid r_j, prmp \mid \Sigma w_j C_j$ (parallel machines)

$1 \mid s_{jk} \mid C_{max}$ (sequence dependent setup / traveling salesman)

$Q2 \mid prec \mid \Sigma w_j T_j$ (2 machines w. different speed, precedence constraints, weighted tardiness)

8

Single Machine Deterministic Models

Jobs: $J_1, J_2, \ldots, J_n$

Assumptions:

- The machine is always available throughout the scheduling period.
- The machine cannot process more than one job at a time.
- Each job must spend on the machine a prescribed length of time.

9

Optimal Solution $1 \| \sum w_j C_j$

- Lets say we have
  - Single machine (1), where
  - the total weighted completion time should be minimized ($\Sigma w_j C_j$)
- **Theorem**: *Weighted Shortest Processing* time first - called the WSPT rule -

$$\frac{w_j}{p_j} \geq \frac{w_k}{p_k}$$

implies job j is sequenced before job k

- *Note: The SPT rule starts with the job that has the shortest processing time, moves on the job with the second shortest processing time, etc.*

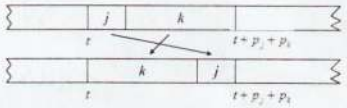
10

Proof (by contradiction)

- Suppose it is not true and schedule S is optimal
- Then there are two adjacent jobs, say job j followed by job k such that

$$\frac{w_j}{p_j} < \frac{w_k}{p_k}$$

- Do a pairwise interchange to get schedule S'



11

Proof (continued)

The weighted completion time of the two jobs under S is

$$(t + p_j)w_j + (t + p_j + p_k)w_k$$

The weighted completion time of the two jobs under S' is

$$(t + p_k)w_k + (t + p_j + p_k)w_j$$

Now:

$$\begin{aligned}(t + p_j)w_j + (t + p_j + p_k)w_k &= (t + p_j)w_j + p_j w_k + (t + p_k)w_k \\ &> (t + p_j)w_j + p_k w_j + (t + p_k)w_k \\ &= (t + p_k)w_k + (t + p_k + p_j)w_j\end{aligned}$$

Contradicting that S is optimal.

12

WSPT Example

Find all optimal sequences for the scheduling problem $1 \parallel \sum w_j C_j$ with the following jobs.

jobs	1	2	3	4	5	6	7
w_j	0	18	12	8	8	17	16
p_j	3	6	6	5	4	8	9

13

WSPT Example Answer

jobs	1	2	3	4	5	6	7
w_j	0	18	12	8	8	17	16
p_j	3	6	6	5	4	8	9
w_j/p_j	0	3	2	1.6	2	2.125	1.78

jobs	2	6	3	5	7	4	1
w_j	18	17	12	8	16	8	0
C_j	6	14	20	24	33	38	41

$\sum w_j C_j = 1610$

or

jobs	2	6	5	3	7	4	1
w_j	18	17	8	12	16	8	0
C_j	6	14	18	24	33	38	41

$\sum w_j C_j = 1610$

14

单机器多条链 $\sum w_j C_j$

$1 \mid chain \mid \sum w_j C_j$

chain 1: $1 \rightarrow 2 \rightarrow \ldots \rightarrow k$
chain 2: $k+1 \rightarrow k+2 \rightarrow \ldots \rightarrow n$

Lemma. If $\frac{\sum_{j=1}^{k} w_j}{\sum_{j=1}^{k} p_j} > \frac{\sum_{j=k+1}^{n} w_j}{\sum_{j=k+1}^{n} p_j}$

the chain of jobs $1,\ldots,k$ precedes the chain of jobs $k+1,\ldots,n$.

Let l^* satisfy $\left(\frac{\sum_{j=1}^{l^*} w_j}{\sum_{j=1}^{l^*} p_j}\right) = \max_{1 \le l \le k} \left(\frac{\sum_{j=1}^{l} w_j}{\sum_{j=1}^{l} p_j}\right) = \rho.$

ρ factor of chain $1,\ldots,k$
l^* is the job that determines the ρ factor of the chain

15

$1 \mid chain \mid \sum w_j C_j$

Lemma. If job l^* determines $\rho(1,\ldots,k)$, then there exists an optimal sequence that processes jobs $1,\ldots,l^*$ one after another without interruption by jobs from other chains.

Algorithm

- Whenever the machine is free, select among the remaining chains the one with the highest ρ factor.
- Process this chain up to and including the job l^* that determines its ρ factor.

16

determined by $\inf\left(\frac{w_j}{p_j}\right)$ speed. higher or lower. if.., stop by (j-1).

$1 \mid chain \mid \sum w_j C_j$

Example

chain 1: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$
chain 2: $5 \rightarrow 6 \rightarrow 7$

jobs	1	2	3	4	5	6	7
w_j	6	18	12	8	8	17	18
p_j	3	6	6	5	4	8	10

ρ factor of *chain 1* is determined by job 2: (6+18)/(3+6)=2.67
ρ factor of *chain 2* is determined by job 6: (8+17)/(4+8)=2.08
chain 1 is selected: jobs 1, 2

ρ factor of the remaining part of *chain 1* is determined by job 3: 12/6=2
ρ factor of *chain 2* is determined by job 6: 2.08
chain 2 is selected: jobs 5, 6

17

$1 \mid chain \mid \sum w_j C_j$

ρ factor of the remaining part of *chain 1* is determined by job 3: 2
ρ factor of the remaining part of *chain 2* is determined by job 7: 18/10=1.8
chain 1 is selected: job 3

ρ factor of the remaining part of *chain 1* is determined by job 4: 8/5=1.6
ρ factor of the remaining part of *chain 2* is determined by job 7: 1.8
chain 2 is selected: job 7

job 4 is scheduled last

the final schedule: 1, 2, 5, 6, 3, 7, 4

18

1|prec|hmax. 1→7→6 5→4 ... 无论如何①⑦⑤.

From back. if there are some prec, then delete some.

1. from all jobs that are schedulable.
   pick the job with lowest cost on the last position.
2. continue until all jobs are scheduled.

eg. jobs 1 2 3
Pi 2 3 5
$h_j(C_j)$ $1+C_j$ $1.5C_2$ 10.
Cmax = 10. last one is job3.
h(job1) = 11
h(job2) = 15 ⇒ h(job1) = 6 ✓
h(job3) = 10 ✓ h(job2) = 7.5
second to last is job 1.
Thus 2→1→3.

P and NP problems

- The efficiency of an algorithm for a given problem is measured by the maximum (worst-case) number of computational steps needed to obtain an optimal solution as a function of the size of the instance.
- Problems which have a known polynomial algorithm are said to be in class P.
- These are problems for which an algorithm is known to exist and it will stop on the correct output while effort is bounded by a polynomial function of the size of the problem.
- For NP (non-deterministic polynomial problems) no simple algorithm yields optimal solutions in a limited amount of computer time.

19

"Hard" Single Machine Problems

- Polynomial time algorithms for more complex precedence constraints than the *simple chains* are developed.
- The problems with *arbitrary* precedence relation are NP hard.
- $1 \mid r_j, prmp \mid \sum w_j C_j$ preemptive version of the WSPT rule does not always lead to an optimal solution, the problem is NP hard
- $1 \mid r_j, prmp \mid \sum C_j$ preemptive version of the SPT rule is optimal
- $1 \mid r_j \mid \sum C_j$ *is NP hard*

20

Lateness Models

- $1 \mid\mid L_{max}$ is the special case of the $1 \mid prec \mid h_{max}$ where $h_j = C_j - d_j$
  h_j are nondecreasing cost functions
  $h_{max} = \max (h_1(C_1), \ldots, h_n(C_n))$
  - Algorithm results in the schedule that orders jobs in increasing order of their due dates - earliest due date first rule (EDD)
- $1 \mid r_j \mid L_{max}$ is NP hard , branch-and-bound is used
- $1 \mid r_j, prec \mid L_{max}$ similar branch-and-bound

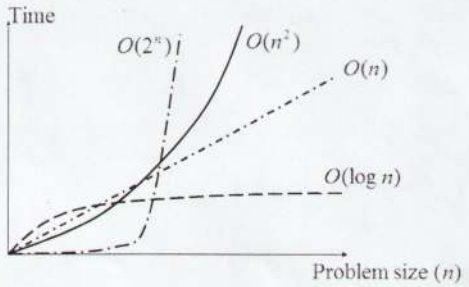
21

Hard vs Easy

- Easy:
  - Sort *n* numbers
  - Solve a system of linear equations
- Hard:
  - Schedule a factory, deliver packages, schedule buses
- $f(n)$: the number of "basic operations" needed to solve the problem with input size *n*
- Easy: $f(n)$ is polynomial in *n*
  - $O(n)$, $O(n \log n)$, $O(n^2)$, ...
- Hard: $f(n)$ is exponential in *n*
  - $O(2^n)$, ...
- The *complexity* of an algorithm is its running time in terms of the input parameters (e.g., number of jobs and number of machines)

22

Polynomial versus NP-Hard



23

Hard vs Easy

$O(n)$	$O(n \log n)$	$O(n^2)$	$O(2^n)$
1	0	1	2
10	10	100	1024
20	26	400	1048576
50	85	2500	1,125,899,906,842,624
100	200	10000	1.268×10^{30}
1000	3000	1,000,000	1.072×10^{301}

24

单机器 release time 有限制.
→ min(L_{max}).

Hard vs Easy

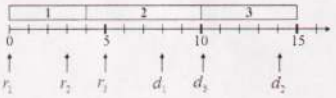
- 10^{301} operations required in worst case
- Age of universe: 10^{18} seconds
- Fastest Computer today: 10^{14} op/sec
- Let's say we get a computer 10^{18} times faster (a sextillion times faster)
  - 10^{33} op/sec
- It may still take 10^{250} times longer than the age of the *universe* to solve the problem!

25

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

Jobs	1	2	3
p_j	4	6	5
r_j	0	3	5
d_j	8	14	10

EDD



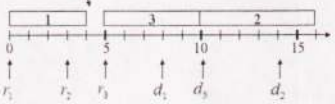
$L_{max} = \max\{L_1, L_2, L_3\} = \max\{C_1 - d_1, C_2 - d_2, C_3 - d_3\}$
$= \max\{4-8, 10-14, 15-10\} = \max\{-4, -4, 5\} = 5$

Can we improve?

26

Delay Schedule

Add a delay



$L_{max} = \max\{L_1, L_2, L_3\} = \max\{C_1 - d_1, C_2 - d_2, C_3 - d_3\}$
$= \max\{4-8, 16-14, 10-10\} = \max\{-4, 2, 0\} = 2$

What makes this problem hard is that the optimal schedule is not necessarily a "non-delay" schedule

27

Final Classic Result

- The preemptive EDD rule is optimal for the preemptive (*prmp*) version of the problem

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

- *Note that in the previous example, the preemptive EDD rule gives us the optimal schedule*

28

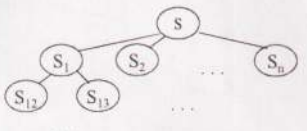
Branch and Bound

- The problem $1 \mid r_j \mid L_{max}$ cannot be solved using a simple dispatching rule so we will try to solve it using branch and bound
- To develop a branch and bound procedure:
  - Determine how to branch
  - Determine how to bound
- Enumerative method
- Guarantees finding the best schedule
- Basic idea:
  - Look at a set of schedules
  - Develop a bound on the performance
  - Discard (fathom) if bound worse than best schedule found before

29

Branch-and-bound algorithm

- Search space can grow very large as the number of variables in the problem increases!
- Branch-and-bound is a methodology that works on the idea of successive partitioning of the search space.

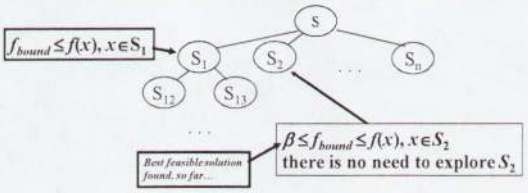


$S = S_1 \cup S_2 \cup \ldots \cup S_n$
$S_1 \cap S_2 \cap \ldots \cap S_n = \varnothing$

30

Branch-and-bound algorithm

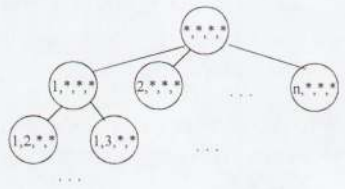
❖ We need some means for obtaining a lower bound on the cost for any particular solution (the task is to minimise the cost).



31

Branch-and-bound algorithm for 1 | r_j | L_{max}

❖ Solution space contains $n!$ schedules (n is number of jobs).
Total enumeration is not viable !



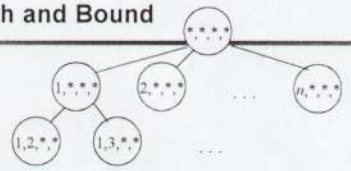
32

Data

Jobs	1	2	3	4
p_j	4	2	6	5
r_j	0	1	3	5
d_j	8	12	11	10

33

Branch and Bound



Branching rule:

❖k level, $j_1, \ldots, j_k$ are scheduled,

❖A job j* needs to be considered if no job still to be scheduled can not be processed before the release time of job j*

that is:

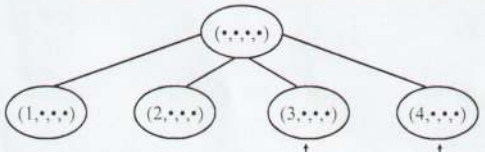
$$r_{j^*} < \min_{l \in J} (\max(t, r_l) + p_l)$$

J set of jobs not yet scheduled

t is time when j_k is completed

34

Branching

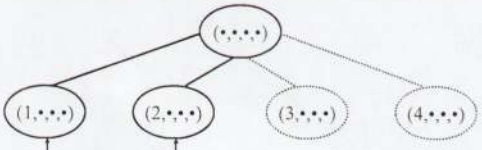


Jobs	1	2	3	4
p_j	4	2	6	5
r_j	0	1	3	5
d_j	8	12	11	10

Discard immediately because
$r_3 = 3$
$r_4 = 5$

35

Branching



Need to develop **lower bounds** on these nodes and do further branching.

36

Bounding (in general)

- Typical way to develop bounds is to relax the original problem to an easily solvable problem
- Three cases:
  - If there is no solution to the relaxed problem there is no solution to the original problem
  - If the optimal solution to the relaxed problem is feasible for the original problem then it is also optimal for the original problem
  - If the optimal solution to the relaxed problem is not feasible for the original problem it provides a bound on its performance

37

Relaxing the Problem

- The problem $1|r_j, prmp|L_{max}$ is a relaxation to the problem $1|r_j|L_{max}$
- Not allowing preemption is a constraint in the original problem
  - **but not the relaxed problem**
- We know how to solve the relaxed problem
  - **preemptive EDD rule**
- Preemptive EDD rule *optimal* for the preemptive version of the problem
- Solution obtained is a *lower bound* on the maximum delay
- If preemptive EDD results in a non-preemptive schedule *all nodes with higher lower bounds can be discarded.*

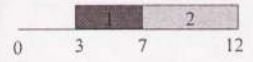
38

each branch has a lower bounds.
choose the branch with lowest one.

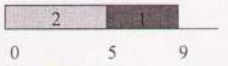
Example

Jobs	1	2
p_j	4	5
r_j	3	0
d_j	4	6

- Non-preemptive schedules

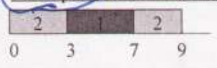


L_1=3
L_2=6
L_{max}=6



L_1=5
L_2=-1
L_{max}=5

- Preemptive schedule obtained using EDD



L_1=3
L_2=3
L_{max}=3

the lowest L_{max}!

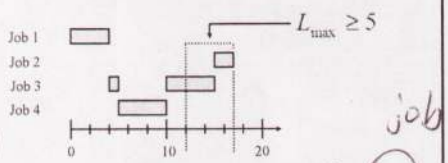
39

按 d_j 排. 满足 r_j 条件 开始.

Lower Bounds

Jobs	1	2	3	4
p_j	4	2	6	5
r_j	0	1	3	5
d_j	8	12	11	10

- Start with (1,•,•,•):
  - Job with EDD is Job 4 but $r_4 = 5$
  - Second earliest due date is for Job 3



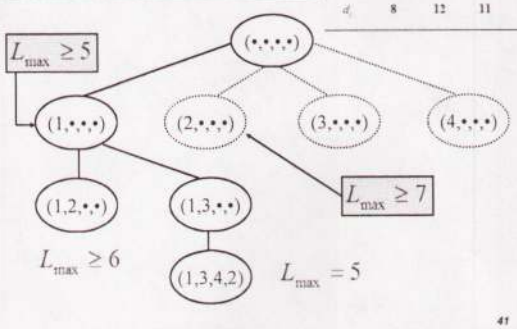
job d_k

job

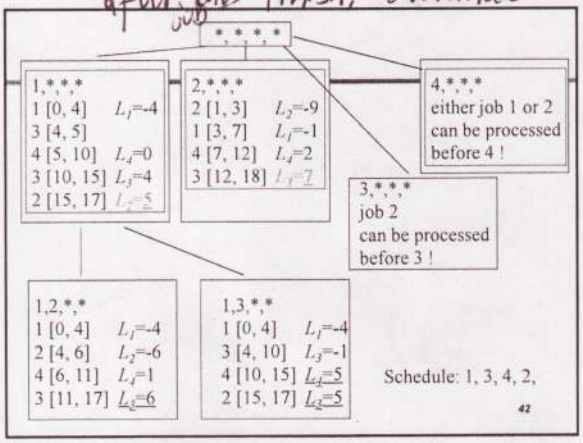
不满足 r_k 时, 按 d_j 排 ... 到 r_k 到, d_k 开始, d_f stop → left after d_k finish, continue.

Branching

Jobs	1	2	3	4
p_j	4	2	6	5
r_j	0	1	3	5
d_j	8	12	11	10



41



42

relaxation, 逐步缩小 relaxation 至逼近 optimum.

Larger Example

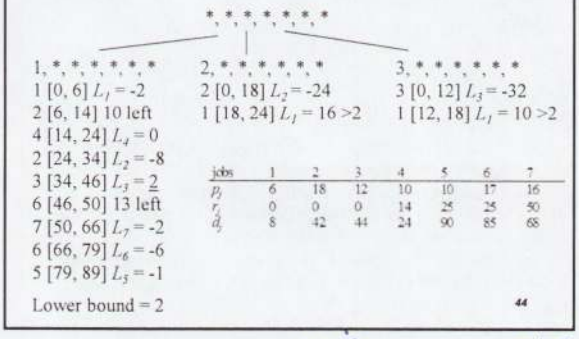
Solve by branch-and-bound the scheduling problem $1 \mid r_j \mid L_{max}$ with the following job data:

jobs	1	2	3	4	5	6	7
p_j	6	18	12	10	10	17	16
r_j	0	0	0	14	25	25	50
d_j	8	42	44	24	90	85	68

43

Solution...

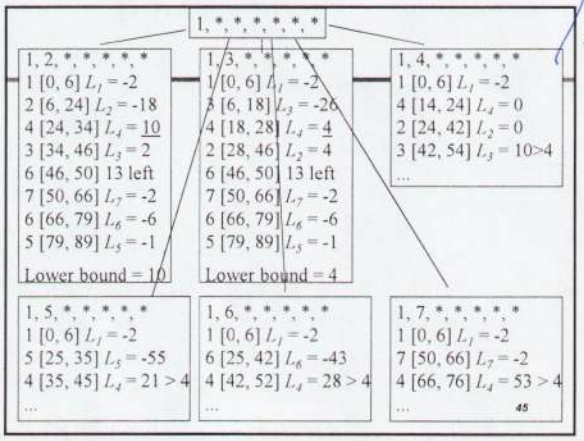
jobs ordered by EDD: 1, 4, 2, 3, 7, 6, 5



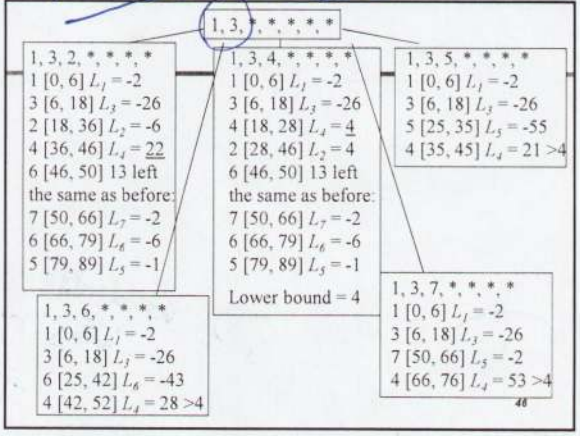
jobs	1	2	3	4	5	6	7
p_j	6	18	12	10	10	17	16
r_j	0	0	0	14	25	25	50
d_j	8	42	44	24	90	85	68

44

注：下表前，先做完。后面都有 preemptive EDD。

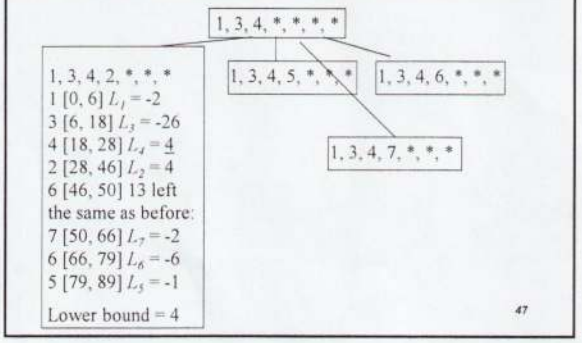


45



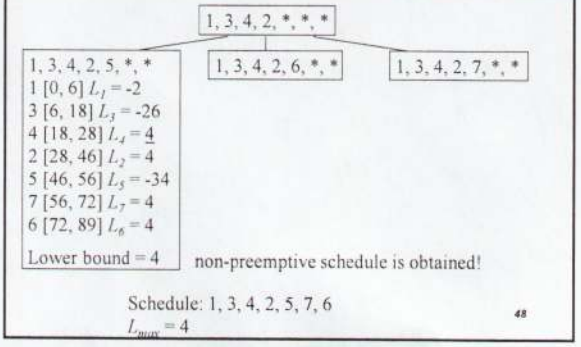
46

Solution...



47

Solution...



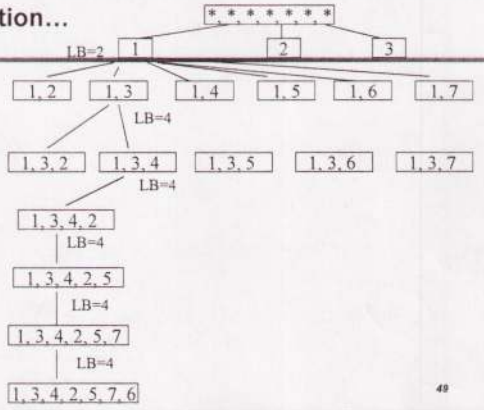
non-preemptive schedule is obtained!

Schedule: 1, 3, 4, 2, 5, 7, 6

L_{max} = 4

48

Solution...



49

Beam Search
Appendix C.4

faster.

- Branch and Bound:
  - Considers every node
  - Guarantees optimum
  - Usually too slow
- Beam search
  - Considers only most promising nodes
  - (beam width)
  - Does not guarantee convergence to optimal solution
  - Much faster

50

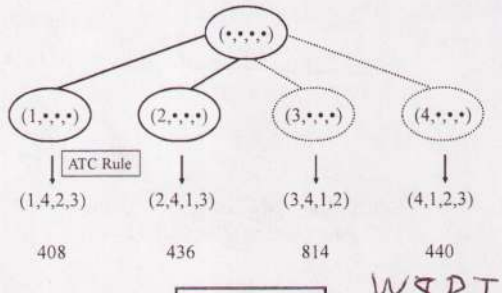
Single Machine Example
(Total Weighted Tardiness)

这里 Σ to w_j, not min (L_{max}).

Jobs	1	2	3	4
p_j	10	10	13	4
d_j	4	2	1	12
w_j	14	12	1	12

51

Branching *(Beam width = 2)*



ATC Rule : Appendix C.3

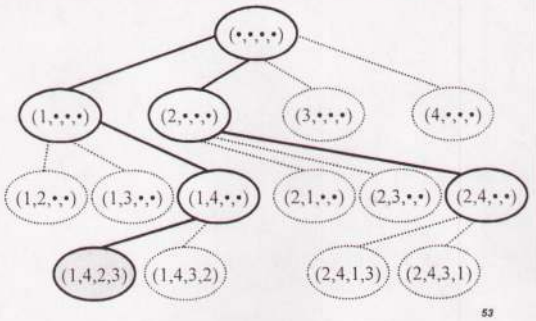
52

WSPT + MS

Apparent Tardiness Cost.

$$I_j(t) = \frac{w_j}{p_j} \exp\left(-\frac{\max(d_j - p_j - t, 0)}{K\bar{p}}\right).$$

Beam Search



53

Discussion

- Implementation tradeoff:
  - Careful evaluation of nodes (accuracy)
  - Crude evaluation of nodes (speed)
- Two-stage procedure
  - Filtering
    - » crude, but fast, evaluation
    - » filter width = set of nodes having the most promise
    - » filter width > beam width
  - Careful evaluation of remaining nodes in the "beam"

54

单机器 单生产线 $\min(\sum w_j T_j)$

$T_j = \max(C_j - d_j, 0)$.

Total Weighted Tardiness

- Problem: $1 \| \sum w_j T_j$
- No efficient algorithm (NP-Hard)
- Branch and bound can only solve small problems (<100 jobs)
- Are there any special cases we can solve?

55

Case 1: *Tight Deadlines*

- Assume $d_j = 0$
- Then

$$T_j = \max\{0, C_j - d_j\} = \max\{0, C_j\} = C_j$$

$$\sum w_j T_j = \sum w_j C_j$$

- We know that WSPT is optimal for this problem!
- The WSPT is optimal in the extreme case and *should be a good heuristic whenever due dates are tight*

56

Case 2: "Easy" Deadlines

- **Theorem**: If the deadlines are sufficiently spread out then the Minimum Slack (MS) rule

$$j_{select} = \arg\min_j \{\max\{d_j - p_j - t, 0\}\}$$

is optimal (*proof a bit harder*)

不证明了.

ATC中用MS

- Conclusion: The MS rule should be a good heuristic whenever deadlines are widely spread out

57

Apparent Tardiness Cost (ATC) Dispatching Rule

ATC Rule : Appendix C.3

- Two good heuristics
  - Weighted Shortest Processing Time *(WSPT)*
    - » optimal with due dates zero
  - Minimum Slack *(MS)*
    - » Near optimal when due dates are "spread out"
  - Any real problem is somewhere in between
- *ATC* combines the characteristics of these rules into one composite dispatching rule

58

Apparent Tardiness Cost (ATC) Dispatching Rule

- Dynamic ranking index

$$I_j(t) = \frac{w_j}{p_j} \exp\left(-\frac{\max\{d_j - p_j - t, 0\}}{K \cdot \bar{p}(t)}\right)$$

Scaling constant

average p_j of the remaining jobs (rounded down)

权衡延迟和成本

- When machine becomes free:
  - Compute index for all remaining jobs that are currently waiting
  - Select job with highest value

highest chosen.

59

Apparent Tardiness Cost (ATC) Dispatching Rule

Special Cases

- If K is very large:
  - ATC reduces to WSPT
- If K is very small and no overdue jobs:
  - ATC reduces to MS
- If K is very small and many overdue jobs:
  - ATC reduces to WSPT applied to overdue jobs

60

Apparent Tardiness Cost (ATC) Dispatching Rule

Choosing *K*

- Value of *K* determined empirically
- Related to the *due date tightness* factor

$$\tau = 1 - \frac{\bar{d}}{C_{\max}}$$

about $\bar{x}$

and the *due date range* factor

$$R = \frac{d_{\max} - d_{\min}}{C_{\max}}$$

about range

极差.

全距.

Apparent Tardiness Cost (ATC) Dispatching Rule

- Usually $1.5 \le K \le 4.5$

$$K = 4.5 + R, \text{ for } R \le 0.5$$
$$K = 6 - 2R, \text{ for } R > 0.5$$

- Rules of thumb:
  - Fix *K=2* for single machine or flow shop.
  - Fix *K=3* for dynamic job shops.

直接用K 不须用R算K.

- Adjusted to reduce weighted tardiness cost in extremely slack or congested job shops
- Statistical analysis/empirical experience

62

Total Weighted Tardiness

从后向前排.

- Schedules are constructed starting from the end (backwards)
- At level k of the b&b tree, the last k jobs in the schedule have been fixed, n-k jobs remain
- Branching: Create a child node of a node at level k-1 by fixing each remaining job at kth position from the end

63

Total Weighted Tardiness

- Lower bounding:
  - Relax the problem by allowing to split the processing of a job to different time intervals
  - Decompose each job j into p_j jobs of unit processing time
  - Define $x_{jk} = 1$, if one unit of job j is processed in time interval [k-1, k] and 0 otherwise
  - Constraints:

把 jobs 单位化.

折分成单位时间段.

$$\sum_{k=1}^{C_{\max}} x_{jk} = p_j \qquad j = 1,\ldots,n$$

$$\sum_{j=1}^{n} x_{jk} = 1 \qquad k = 1,\ldots,C_{\max}$$

64

Total Weighted Tardiness

- Define cost coefficients c_{jk}

  $c_{jk} = 0$, for $k \le d_j$ and $c_{jk} = w_j$, for $k > d_j$

- Then, $\sum_{k=1}^{C_{\max}} c_{jk} x_{jk} \le w_j T_j$ relaxed
- This is a "transportation problem" and can be solved in polynomial time to obtain a lower bound
- Bounding scheme applied to all unscheduled jobs at a node

Some of the partial solutions can be eliminated based on this observation:

If for two jobs j and k, $d_j \le d_k$, $p_j \le p_k$, and $w_j \ge w_k$, then there exists an optimal solution in which j appears before k

obviously.

65

Total Weighted Tardiness

初步筛选

If for two jobs j and k,

$p_j \le p_k$,
$d_j \le d_k$,
$w_j \ge w_k$,

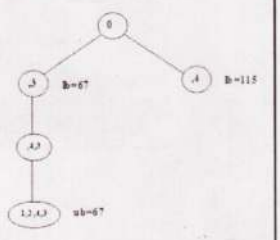
and then there exists an optimal solution in which j appears before k

Numeric Example:

Jobs	1	2	3	4
w_j	4	5	3	5
p_j	12	8	15	9
d_j	16	26	25	27

Compare jobs 2 and 4: 2 must come before 4
Compare jobs 1 and 3: 1 must come before 3

- *Only jobs 3 and 4 can be last*



Total Weighted Tardiness

Calculation of the lower bound at node (-,4) :

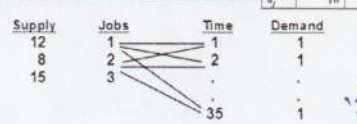
- Solve a transportation problem with jobs 1, 2, 3 and costs below

$c_{1k} = 0, \quad k = 1,\ldots,16$	$c_{1k} = 4, \quad k = 17,\ldots,35$
$c_{2k} = 0, \quad k = 1,\ldots,26$	$c_{2k} = 5, \quad k = 27,\ldots,35$
$c_{3k} = 0, \quad k = 1,\ldots,25$	$c_{3k} = 3, \quad k = 26,\ldots,35$

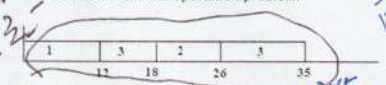
67

Total Weighted Tardiness

Jobs	1	2	3	4
w_j	4	5	3	5
p_j	12	8	15	9
d_j	16	26	25	27



Solution of the transportation problem:



LB = 3*10 + $w_4 T_4$ = 30 + 85 = 115

- After the exploration of all active b&b nodes, it can be verified that
  1, 2, 4, 3 is the best overall schedule

68

Total Earliness and Tardiness

- definition: Earliness of job i = E_i = max { $d_i - C_i$, 0 }
- Objective is to minimize total Earliness and Tardiness

$$\sum_{i=1}^{n} E_i + \sum_{i=1}^{n} T_i$$

- *Not* a regular measure of performance
  - if C_i decreases, earliness could *increase*
  - leads to the possibility of deliberately inserting idle time
  - minimizing earliness tends to minimize work-in-process inventory, a core objective in Just-in-Time production scheduling

69

Total Earliness and Tardiness
Common Due Date

- all $d_i = d$
- no inserted idle time
  - once processing begins, *jobs continually scheduled*
- Two sets of jobs can be defined:
  - J_1 : all jobs with completion times ≤ d
  - J_2 : all jobs with completion times > d
- Optimal sequencing rule:
  - Sequence J_1 using WLPT $\quad \frac{w_1'}{p_1} \le \frac{w_2'}{p_2} \le \ldots$
  - Sequence J_2 using WSPT $\quad \frac{w_1''}{p_1} \ge \frac{w_2''}{p_2} \ge \ldots$

70

Total Earliness and Tardiness
Common "Loose" Due Date

- If d is large enough such that no job begins at t = 0,
  - an *optimal schedule exists* with a job i with $C_i = d$
- If all earliness and tardiness penalties = 1,
  - order all jobs using LPT
  - assign first job from list to J_1, next job to J_2, and alternate thereafter until all jobs have been assigned
  - J_1 already in LPT order; reverse the order of jobs in J_2 to achieve SPT ordering
- Example (with all weights = 1):

job i	1	2	3	4	5
p_i	5	7	4	8	12
LPT order	4	3	5	2	1
k:J_k	2	1	1	2	1

LPT ordered J_1 : 5-2-3
SPT ordered J_2 : 1-4

71

Total Earliness and Tardiness
Common "Loose" Due Date

- Job *b* in the optimal sequence is completed at the due date where *b* is the smallest integer satisfying

$$\sum_{i \in J_1} (w_i' + w_i'') \ge \sum_{i=1}^{b} w_i''$$

- Example:
  d = 50

$$\sum_{i=1}^{n} w_i'' = 53$$

job i	1	2	3	4	5
p_i	5	7	4	8	12
w_i'	4	8	8	12	15
w_i''	7	7	4	20	15

- order by non-increasing $w_i' + w_i''$
  4,5,2,3,1

b	2	2
J_1 sum(1,b)	32	62

- WLPT J_1 : 5-4
- WSPT J_2 : 1-2-3 or 1-3-2

72

若放入 d^+，则使下一个放入 d^+ 的 job min(Tardiness).
若放入 d^-，则使其 min(Earliness).

Tight

Total Earliness and Tardiness
Common "Tight" Due Date

- "tight" due date implies a job *must* begin at t = 0
  - an optimal sequence no longer has a job completing at t = d and no optimal rule-based procedure exists
- Heuristic procedure for constructing sequence (no earliness or tardiness penalties)

0. LPT order jobs first
1. set $t_1 \leftarrow d$, $t_2 \leftarrow \Sigma(p_j)-d$, $k \leftarrow 1$
2. *if* $t_1 > t_2$ *then* assign job to earliest available slot $t_1 \leftarrow t_1 - p_k$
   else assign job to latest available slot $t_2 \leftarrow t_2 - p_k$
3. *if* $k<n$ *then* $k \leftarrow k+1$, go to (2)
   else STOP

then

73

Total Earliness and Tardiness
Common "Tight" Due Date

- Example: d = 15

job i	1	2	3	4	5
p_i	5	7	4	8	12
LPT order k	4	3	5	2	1

k	t_1	t_2	assignment	partial seq.
1	15	36-15=21	15<21→job 5 pushed late	{*,*,*,*,5}
2	15	21-12=9	15>9→job 4 pushed early	{4,*,*,*,5}
3	15-8=7	9	7<9→job 2 pushed late	{4,*,*,2,5}
4	7	9-7=2	7>2→job 1 pushed early	{4,1,*,2,5}
5				**{4,1,3,2,5}**

74

注：已 d^- 把 Σp_j 分成 d^- 和 d^+ 两部分.
把 remaining job 中 p_j 最大的放到 $\max(d^-, d^+)$.

Total Earliness and Tardiness
Distinct Due Dates

- Common due date properties do not apply *(NP-hard)*
- May need to insert idle time to further minimize objective
- Two-stage scheduling heuristic
  - Build a sequence
  - Determine start times for jobs in the sequence
- Property: If $d_{j+1} - d_j \le p_{j+1}$ → no idle time between j and j+1
- A *cluster* of jobs [u,v] is a series of jobs starting with j = u-1 and ending with j = v-1 for which the *no idle time property* holds
- Idle time insertion algorithm starts with a sequence with no idle time
- Algorithm determines the *optimal shift* of each cluster to minimize E/L objective for a given sequence

75

E/T.

Local (Neighborhood) Search

Step. 1. Initialisation
k=0
Select a starting solution $S_0 \in S$
Record the current best-known solution by setting $S_{best} = S_0$ and $best\_cost = F(S_{best})$

Step 2. Choice and Update
Choose a Solution $S_{k+1} \in N(S_k)$ - near S_k.
If the choice criteria cannot be satisfied by any member of $N(S_k)$, then the algorithm stops
if $F(S_{k+1}) < best\_cost$ then $S_{best} = S_{k+1}$ and $best\_cost = F(S_{k+1})$

Step 3. Termination
If termination conditions apply
then the algorithm stops
else $k = k+1$ and go to Step 2.

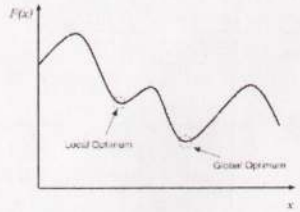
76

Distinct

actually $\min(\Sigma T_i)$

Local (Neighborhood) Search

- ***Global Optimum***: better than all other solutions
- ***Local Optimum***: better than all solutions in a certain *neighborhood*



77

Neighborhood Search

- Seeks *local* (myopic) optimal solution
- Initial n job *seed* sequence:
  - at random
  - solution of appropriate heuristic
- Generate solutions in *neighborhood* of seed
  - API: n-1 API's describe neighborhood
  - TPUF: "top priority up front"
    - » TPUF neighborhood for seed 1-2-3-4 :
      - 2-1-3-4
      - 3-1-2-4
      - 4-1-2-3

Adjacent Pairwise Interchange

78

Neighborhood Search

- PI: General pairwise interchange
  - » all (n)(n-1)/2 possible sequences generated
  - » PI neighborhood for seed 1-2-3-4 :
    - 2-1-3-4
    - 3-2-1-4
    - 4-2-3-1
    - 1-3-2-4
    - 1-4-3-2
    - 1-2-4-3
- K-move: move K jobs to left (or right) *at once*
  - » (2K-1)n sequences generated
  - » about as accurate as PI with 2K/n the computational cost
  - » TPUF is K-move with K=1

79

Neighborhood Search

- k-way interchange
  - » interchange among every possible subset of k jobs
  - » n!/(k!(n-k)!) sequences
  - » n-way is *complete enumeration*
  - » k>3 becomes too costly to be practical
- Selecting next seed
  - evaluate all solutions in neighborhood and choose best
  - choose first improved solution... *usually* more efficient
- Terminating neighborhood search
  - no (or small) improvement over neighborhood
  - no (or small) improvement over *expanded* neighborhood
    - » after having gone from API to PI, for example

80

Adjacent Pairwise Interchange (API)

Minimize Total Tardiness

- Use EDD rule to provide "seed" sequence to minimize total tardiness
- Examine API neighborhood of seed
- First improved sequence becomes the next seed
- Example: (min total tardiness = 755)

Job i	1	2	3	4	5	6	7	8
p_i	121	147	102	79	130	83	96	88
d_i	260	269	400	266	337	336	683	719

81

Adjacent Pairwise Interchange (API)

- Calculating total tardiness for API generated sequences

S	T_i	S	T_i	S	T_i	S	T_i	S	T_i	S	T_i	S	T_i	S	T_i
1	0	4	0	1	0	1	0	1	0	1	0	1	0	1	0
4	0	1	0	2	0	4	0	4	0	4	0	4	0	4	0
2	78	2	78	4	81	6	0	2	78	2	78	2	78	2	78
6	94	6	94	6	94	2	161	5	140	6	94	6	94	6	94
5	223	5	223	5	223	5	223	6	224	3	132	5	223	3	223
3	262	3	262	3	262	3	262	3	262	5	325	7	0	5	262
7	75	7	75	7	75	7	75	7	75	7	75	3	358	8	31
8	127	8	127	8	127	8	127	8	127	8	127	8	127	7	163
Σ T	859		859		862		*848*		906		*831*		880		851

-11 -28 -8

14 02 35 87

Adjacent Pairwise Interchange (API)

- First improved sequence is seed for another API iteration

								Σ T
1	4	6	2	5	3	7	8	848
4	1	6	2	5	3	7	8	848
1	6	4	2	5	3	7	8	865
1	4	2	6	5	3	7	8	*781*

83

Adjacent Pairwise Interchange (API)

- Another total tardiness example...

Job i	1	2	3	4	5	6	7	8
p_i	138	132	153	89	141	131	107	103
d_i	466	459	402	422	478	392	368	385

EDD Seq									Σ T
	7	8	6	3	4	2	1	5	1412
	8	7	6	3	4	2	1	5	1412
	7	6	8	3	4	2	1	5	1412
	7	8	3	6	4	2	1	5	1422
	7	8	6	4	3	2	1	5	*1348*

84

Adjacent Pairwise Interchange (API)

new seed seq									Σ T
	7	8	6	4	3	2	1	5	1348
	8	7	6	4	3	2	1	5	1348
	7	6	8	4	3	2	1	5	1348
	7	8	4	6	3	2	1	5	1378
	7	8	6	3	4	2	1	5	1412
	7	8	6	4	2	3	1	5	*1327*

min Σ T = 1300

85

Neighborhood Search Example

- Investigating neighborhood search strategies to minimize total tardiness

Job i	1	2	3	4	5	6	7	8
p_i	138	132	153	89	141	131	107	103
d_i	466	459	402	422	478	392	368	385

- Seed generated using Longest Processing Time first rule (LPT)... iteration 1

Seq	3	5	1	2	6	7	8	4	
C_i	153	294	432	564	695	802	905	994	Σ T_i
T_i	0	0	0	105	303	434	520	572	1934

86

初始 rule 不重要. (LPT, EDD).
反正要 improve.

Neighborhood Search Example

- API... iteration 2

Seq	3	5	1	2	*7*	*6*	8	4	
C_i	153	294	432	564	*671*	*802*	905	994	Σ T_i
T_i	0	0	0	105	*303*	*410*	520	572	1910

- API... iteration 3

Seq	3	5	1	2	7	*8*	*6*	4	
C_i	153	294	432	564	671	*774*	*905*	994	Σ T_i
T_i	0	0	0	105	303	*389*	*513*	572	1882

87

Neighborhood Search Example

- API... iteration 8

Seq	3	7	5	1	8	2	4	6	
C_i	153	260	401	539	642	774	863	994	Σ T_i
T_i	0	0	0	73	257	315	441	602	1688

- API... iteration 20

Seq	3	7	8	4	6	2	1	5	
C_i	153	260	363	452	583	715	853	994	Σ T_i
T_i	0	0	0	30	191	256	387	516	1380

- min total tardiness = 1300

88

Neighborhood Search Example

- Using PI with same LPT seed at iteration 1
  - iteration 2:

Seq	3	*4*	1	2	6	7	8	*5*	
C_i	153	242	380	512	643	750	853	994	Σ T_i
T_i	0	0	0	53	251	382	468	516	1670

- PI... iteration 6

Seq	7	6	8	4	2	3	1	5	
C_i	107	238	341	430	562	715	853	994	Σ T_i
T_i	0	0	0	8	103	313	387	516	1327

89

Neighborhood Search Example

- PI... iteration 9

Seq	7	6	8	4	2	1	5	3	
C_i	107	238	341	430	562	700	841	994	Σ T_i
T_i	0	0	0	8	103	234	363	592	1300

  - ***Optimal Solution***

- More Sophisticated Neighborhood Search Approaches:
  - Simulated Annealing (App. C.5)
  - Tabu Search (App. C.5
  - Genetic Algorithms (App. C.6)

90

Construction versus Improvement Heuristics

- Construction Heuristics
  - Start without a schedule
  - Add one job at a time
  - Dispatching rules and beam search
- Improvement Heuristics
  - Start with a schedule
  - Try to find a better 'similar' schedule
- Can be combined

91

Backward-Forward Heuristic (BF)

D.R. Sule, *Industrial Scheduling,* PWS Publishing Co., Boston, 1997.

- Backward phase : construct an initial sequence, starting with the last position in the sequence.
- Forward phase : attempt to improve the solution by moving later jobs in the sequence *forward* to earlier positions.
- Backward phase notation:
  - N = number of jobs
  - S = set of sequenced jobs
  - S^c = set of unsequenced jobs
  - p_i = processing time for job i
  - d_i = due date for job i
  - w_i'' = tardiness penalty for job i

92

Backward-Forward Heuristic (BF)

- Backward phase algorithm:

FOR n = N, N-1, N-2,..., 1

SET $t = \sum_{i \in S^c} p_i$

P_i = tardiness penalty for unsequenced job i = $\max\{(t-d_i)w_i'', 0\}$

FIND i* = job to sequence in position n = $\arg\min_{i \in S^c}\{P_i\}$ ...largest p_i is tie-breaker

$S^c = S^c / \{i^*\}$

ENDFOR

OUTPUT: S

93

Backward-Forward Heuristic (BF)

- Backward phase example:

job i	1	2	3	4
p_i	2	4	3	1
d_i	1	2	4	6
w_i''	1	1	1	1

- Iterations of algorithm:

n	S^c	t	P_i, $i \in S^c$	i*
4	1,2,3,4	10	9,8,6,4	4
3	1,2,3	9	8,7,5	3
2	1,2	6	5,4	2
1	1	2	1	1

S=1-2-3-4

$P(S) = \sum_{i \in S} P_i$ = 14

94

Backward-Forward Heuristic (BF)

- Forward phase
  - k = "lag" between job positions in S
  - $J_{[i]}$ = job in position i in sequence
  - Algorithm:
    - **(1)** SET k = N-1
    - **(2)** SET j = k+1
    - **(3)** SWAP $J_{[j]}$ and $J_{[j+k]}$ to form S'
      FIND P(S')
      IF P(S')>P(S), SET j ← j+1
      IF j < N, go to (3)
      ELSE, go to (4)
      ELSE, IF SWAP not done earlier, then S ← S', go to (1)
      ELSE, SET j ← j+1
      IF j < N, go to (3)
      ELSE,
      (4) k ← k-1
      IF k > 0, go to (2)
      ELSE, *STOP* (with S)

95

Backward-Forward Heuristic (BF)

- Example continued with forward phase algorithm

lag k	[j]	[j-k]	S'	$C_{[j]}-d_{[j]}$	P(S')
3	4	1	4-2-3-1	*1-6*, 5-2, 8-4, 10-1	16 > P(S)
2	3	1	3-2-1-4	*3-4*, 7-2, 9-1, 10-6	17 > P(S)
2	4	2	1-4-3-2	2-1, *3-6*, 6-4, 10-2	11 < P(S)

S = 1-4-3-2

96

Backward-Forward Heuristic (BF)

- Continuing the forward algorithm on S = 1-4-3-2

lag k	[j]	[j-k]	S'	$C_{[j]}$-$d_{[i]}$	P(S')
3	4	1	2-4-3-1	4-2, *5-6*, 8-4, 10-1	15>P(S)=11
2	3	1	3-4-1-2	*3-4, 4-6*, 6-1, 10-2	13 > P(S)
2	4	2	1-2-3-4	done earlier..	14 > P(S)
1	2	1	4-1-3-2	*1-6*, 3-1, 6-4, 10-2	12 > P(S)
1	3	2	1-3-4-2	2-1, 5-4, 6-6, 10-2	10 < P(S)

S = 1-3-4-2

97

Backward-Forward Heuristic (BF)

- Continuing the forward algorithm on S = 1-3-4-2

lag k	[j]	[j-k]	S'	$C_{[j]}$-$d_{[i]}$	P(S')
3	4	1	2-3-4-1	4-2, 7-4, 8-6, 10-1	16>P(S)=10
2	3	1	4-3-1-2	*1-6*, 4-4, 6-1, 10-2	13 >P(S)
2	4	2	1-2-4-3	2-1, 6-2, 7-6, 10-4	12 > P(S)
1	2	1	3-1-4-2	*3-4*, 5-1, 6-6, 10-2	12 > P(S)
1	3	2	1-4-3-2	done earlier	11 > P(S)
1	4	3	1-3-2-4	2-1, 5-4, 9-2, 10-6	13 > P(S)

S* = 1-3-4-2 P(S*) = 10

98

Backward-Forward Heuristic (BF)

- Total Earliness and Tardiness Objective
  - w_i' = earliness penalty rate for completing job i before d_i
  - BF algorithm adjustment in backward phase:
    $P_i = \max \{ (t-d_i)w_i'' , (d_i-t)w_i' \}$
- Early and Late due dates
  - DE_i : early penalty applies if $C_i < DE_i$
  - DL_i : late penalty applies if $C_i > DL_i$
  - Backward phase adjustment:
    $P_i = \max \{(DE_i-C_i)w_i' , 0 , (C_i-DL_i)w_i''\}$
    » if several jobs have lowest P_i , use job with longest p_i

99

Parallel Machine Model Characteristics

Identical machines in parallel *Pm*

- *m* machines in parallel
- Job *j* requires a single operation and may be processed on any of the *m* machines
- If job *j* may be processed on any one machine belonging to a given subset *Mj*
  Pm | Mj | ...
- Machines in parallel with different speeds *Qm*
- Unrelated machines in parallel *Rm*
- machines have different speeds for different jobs

100

Parallel Machine Models

- Resource allocation in addition to sequencing
  - Single stage
  - Identical processors assumed (initially...)
  - Independent jobs (initially...)
  - Job processed on one machine at a time (initially...)
- $P_m \mid\mid C_{max}$: Makespan objective without preemption
  - "load balancing" objective
  - Integer Program: $\min C_{max}$
    » $x_{ij} = 1$, if job i assigned to machine j $s.t.$
    $= 0$, if not

$$\sum_{j=1}^{m} x_{ij} = 1 \ , \ i=1,\ldots,n$$

$$\sum_{i=1}^{n} p_i x_{ij} \le C_{max} \ , \ j=1,\ldots,m$$

101

Throughput and Makespan

- Throughput
  - Defined by bottleneck machines
- Makespan

$$C_{max} = \max\{C_1, C_2, \ldots, C_n\}$$

$$C_i = \max\{C_{i1}, C_{i2}, \ldots, C_{im}\}, i = 1,\ldots,n$$

- Minimizing makespan tends to
  - maximize throughput and
  - balance load

102

Pm || Cmax. Parallel Machine Makespan without preemption.

machine j 的 close time.

jobs close time.

machine with job j close time.

Parallel Machine Models

Makespan $P_m \,|\, | \, C_{max}$

- Largest Processing Time heuristic *LPT*
  1. Sequence jobs with LPT rule
  2. In order, assign jobs to machines least loaded
     - » shortest jobs used near end to refine balance among machines
- LPT worst case analysis: $\frac{C_{max}(LPT)}{C_{max}(OPT)} \le \frac{4}{3} - \frac{1}{3m}$

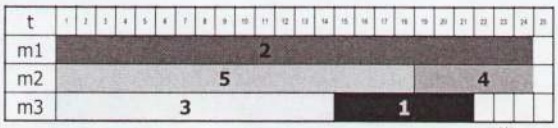
Parallel Machine Models

Makespan $P_m \,|\, | \, C_{max}$

- LPT example:

job i	1	2	3	4	5
p_i	7	24	14	6	18

  - ***LPT order:*** **2-5-3-1-4**
  - **C_{max} = 24**



Parallel Machine Models

Makespan $P_m \,|\, | \, C_{max}$

- MULTIFIT heuristic

Reference: E.G. Coffman, M.R. Garey, D.S. Johnson (1978) "An Application of Bin-Packing to Multiprocessor Scheduling", *SIAM Journal on Computing*, Vol. 7, pp. 1-17

- better worst case bound (k = # of iterations)

$$\frac{C_{max}(MULTIFIT)}{C_{max}(OPT)} \le 1.22 + \left(\frac{1}{2}\right)^k$$

- Define:

$$C_L = \max\left\{p_i, \frac{\sum_{i=1}^{n} p_i}{m}\right\} \text{ and } C_U = \max\left\{p_i, 2\frac{\sum_{i=1}^{n} p_i}{m}\right\}$$

- C_L and C_U adjusted by algorithm
- Estimate of C_{max} = final value of C_U

Parallel Machine Models

Makespan $P_m \,|\, | \, C_{max}$

- MULTIFIT Algorithm:

0. SET I ← 0, LPT order the jobs.
1. SET C ← (1/2)(C_L + C_U)
   I ← I + 1
2. Assign the jobs in succession to the lowest indexed machine that can complete the job within the capacity, C
   - » *First Fit Decreasing (FFD) bin packing heuristic*
3. If all the jobs can be assigned to the m machines, then SET C_U ← C and go to step 4.
   Else, SET C_L ← C and go to step 4.
4. If I = k (prespecified), STOP
   Else, go to step 1.

Parallel Machine Models

Makespan $P_m \,|\, | \, C_{max}$

- MULTIFIT example: m = 3

job i	1	2	3	4	5
p_i	7	24	14	6	18
LPT job sort	2	5	3	1	4
sorted p_i	24	18	14	7	6

C_U = max { 24, 2(69)/3 } = 46
C_L = max { 24, 69/3 } =24

SET I = 0

Parallel Machine Models

Makespan $P_m \,|\, | \, C_{max}$

- *step 1:* SET C = ½[24+46] = 35 , I = 1
- *step 2:*

LPT job i	1	2	3	4	5
i → m_K	1	2	2	1	3
remaining capacity on m_1	**11**	11	11	**4**	4
remaining capacity on m_2	35	**17**	**3**	3	3
remaining capacity on m_3	35	35	35	35	**29**

- *step 3:* C_U = C = 35

job i	1	2	3	4	5
p_i	7	24	14	6	18
LPT job sort	2	5	3	1	4
sorted p_i	24	18	14	7	6

Parallel Machine Models

Makespan $P_m \,//\, C_{max}$

❖ *step 1:* SET C = ½[24+35] = 29.5 , I = 2

❖ *step 2:*

LPT job i	1	2	3	4	5
$i \rightarrow m_K$	1	2	3	2	3
m_1	**5.5**	5.5	5.5	5.5	5.5
m_2	29.5	**11.5**	11.5	**4.5**	4.5
m_3	29.5	29.5	**15.5**	15.5	**9.5**

remaining capacity on m_K

❖ *step 3:* C_U = C = 29.5

job i	1	2	3	4	5
p_i	7	24	14	6	18
LPT job sort	2	5	3	1	4
sorted p_i	24	18	14	7	6

Parallel Machine Models

Makespan $P_m \,//\, C_{max}$

❖ *step 1:* SET C = ½[24+29.5] = 26.75 , I = 3

❖ *step 2:*

LPT job i	1	2	3	4	5
$i \rightarrow m_K$	1	2	3	2	3
m_1	**2.75**	2.75	2.75	2.75	2.75
m_2	26.75	**8.75**	8.75	**1.75**	1.75
m_3	26.75	26.75	**12.75**	12.75	**6.75**

remaining capacity on m_K

❖ *step 3:* C_U = C = 26.75

job i	1	2	3	4	5
p_i	7	24	14	6	18
LPT job sort	2	5	3	1	4
sorted p_i	24	18	14	7	6

Parallel Machine Models

Makespan $P_m \,//\, C_{max}$

❖ *step 1:* SET C = ½[24+26.75] = 25.375 , I = 4

❖ *step 2:*

LPT job i	1	2	3	4	5
$i \rightarrow m_K$	1	2	3	2	3
m_1	**1.375**	1.375	1.375	1.375	1.375
m_2	25.375	**7.375**	7.375	**.375**	.375
m_3	25.375	25.375	**11.375**	11.375	**5.375**

remaining capacity on m_K

❖ *step 3:* C_U = C = 25.375

job i	1	2	3	4	5
p_i	7	24	14	6	18
LPT job sort	2	5	3	1	4
sorted p_i	24	18	14	7	6

Parallel Machine Models

Makespan $P_m \,//\, C_{max}$

❖ *step 1:* SET C = ½[24+25.375] = 24.69 , I = 5

❖ *step 2:*

LPT job i	1	2	3	4	5
$i \rightarrow m_K$	1	2	3	3	2
m_1	**.69**	.69	.69	.69	.69
m_2	24.69	**6.69**	6.69	6.69	**.69**
m_3	24.69	24.69	**10.69**	**3.69**	3.69

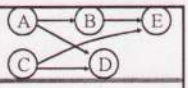
remaining capacity on m_K

❖ *step 3:* C_U = C = 24.69

❖ Same schedule produced for I > 5 *(same as LPT produces)*

Parallel Machine Models

Makespan $P_m \,//\, prec \,//\, C_{max}$



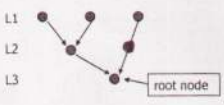
- If the number of parallel machines at least as large as the the number of jobs → classic *Project Scheduling* problem
- Algorithmic approach used to minimize makespan of a project is the *Critical Path Method*
- Forward procedure:
  - Starting at time zero, calculate the **earliest** each job can be started
  - The completion time of the last job is the makespan
- Backward procedure
  - Starting at time equal to the makespan, calculate the **latest** each job can be started so that this makespan is realized

Parallel Machine Models

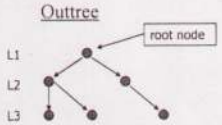
Makespan $P_m \,//\, prec \,//\, C_{max}$

- "Harder" class of parallel machine scheduling problems have $1 < m < n$
- Easy special case: $P_m \,//\, p_j=1 \,,\, tree \,//\, C_{max}$
- Trees:

Intree



Outtree



多机器 有 prec, min (Cmax)

Parallel Machine Models

Makespan P_m / prec / C_{max}

- *Critical Path Rule (CP) :*
  - highest priority to job in front of longest "string" (jobs on highest level first)
  - ties arbitrarily broken
  - optimal scheduling rule for intrees and outtrees with unit processing times
- CP rule used as a heuristic for arbitrary precedence constraints and all processing times equal
  - worst case bound for m=2: *(even worse for m>2...)*

$$\frac{C_{\max}(CP)}{C_{\max}(OPT)} \leq \frac{4}{3}$$

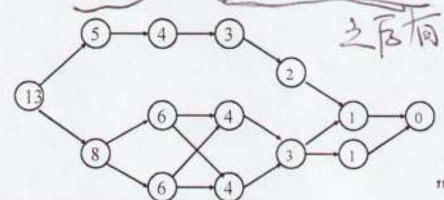
115

Parallel Machine Models

Makespan P_m / prec / C_{max}

- ***Largest Number of Successors first : (LNS)***
  - heuristic for arbitrary precedence and equal processing times
  - optimal for intrees and outtrees (since equivalent to CP, in that case)
  - Example: *nodes labeled with number of successor nodes*

之后有几个 nodes.



116

后面 nodes 最多的先上.

Parallel Machine Models

Makespan P_m / prec / C_{max}

- Applying the LNS rule for arbitrary precedence *and* arbitrary processing times
  - Assign to each terminal job a label equal to its processing time
  - Assign to job j the label β_j ,where → job j 之后

$$\beta_j = \sum_{i \in S(j)} p_i + p_j$$

  - S(j) is the set of all jobs on all paths in the precedence diagram that succeed job j
  - Assign the job with no unscheduled predecessors with the largest label to the least loaded machine

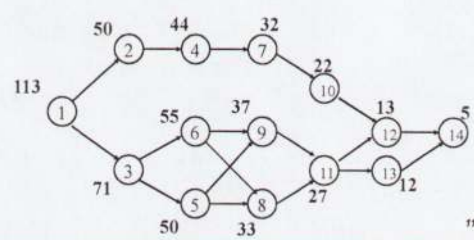
117

→ 最大数的放到最闲的机器上.

Parallel Machine Models

Critical Path Method Example / LNS labeling

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5



118

Parallel Machine Models

Makespan P_m / prec / C_{max}

- Machine loading using LNS labels (m = 3):

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5
LNS Label	113	50	71	44	50	55	32	33	37	22	27	13	12	5
immediate predecessors	—	1	1	2	3	3	4	5,6	5,6	7	6,9	10,11	11	12,13

t		5	11	14		23	26	32	33	36	42	43	50	51	56		
m_1	1				5			8									
m_2			3		6			9			11		12		14		
m_3			2	4			7			10			13				

119

Parallel Machine Models

Makespan P_m / prmp / C_{max}

- *Preemption:* processing of job may be interrupted by beginning to process another job
  - preempt-repeat: preempted job must start over
  - preempt-resume: preempted job resumes processing where it was interrupted (our assumption here...)
- LP formulation of P_m | prmp | C_{max}
  - let x_{ij} = amount of processing time for job j on machine i

$$\min \quad C_{\max}$$

$$s.t.$$

$$\sum_{i=1}^{m} x_{ij} = p_j \quad , \; j=1,\ldots,n$$

$$\sum_{i=1}^{m} x_{ij} \leq C_{\max} \quad , \; j=1,\ldots,n$$ → job 的 Cmax

$$\sum_{j=1}^{n} x_{ij} \leq C_{\max} \quad , \; i=1,\ldots,m$$ → machine 的 Cmax

120

Parallel Machine Models

Makespan P_m / prmp / C_{max}

- Lower bound on C_{max}:

$$C^*_{max} = \max\left\{ p_j, \frac{\sum_{j=1}^{n} p_j}{m} \right\} \le C_{max}$$

- P_m | prmp | C_{max} Algorithm
  1. Load all jobs in LPT order on a single machine
  2. From the beginning of the single machine schedule, allocate C^*_{max} "chunks" to each machine in turn with any residual chunk < C^*_{max} allocated to the last machine
  - *Note:* Since $C_{max} = C^*_{max}$, schedule is *optimal*

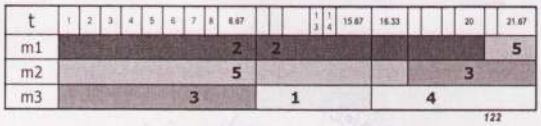
Parallel Machine Models

Makespan P_m / prmp / C_{max}

- P_m | prmp | C_{max} Algorithm Example

job i	1	2	3	4	5
p_i	7	20	14	6	18
LPT job sort	2	5	3	1	4
sorted p_i	20	18	14	7	6

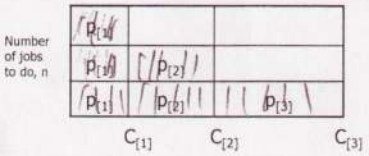
- C^*_{max} = max { 20, 65/3 } = 21.67



Parallel Machine Models

Total Completion Time P_m / / ΣC_j

- SPT rule *optimal* for P_m | | ΣC_j
- Visualizing *min* ΣC_j on a single machine
  - min $\Sigma C_j = n p_{[1]} + (n-1)p_{[2]} + \ldots + p_{[n]}$
  - for n = 3:



  - shaded area is smallest when smallest p_j matched with largest coefficient

Parallel Machine Models

Total Completion Time P_m / / ΣC_j

- Generalizing to m parallel machines:
  - let i[j] = j-th job in sequence on machine i

$$\sum_{j=1}^{n} C_j = \sum_{i=1}^{m} \sum_{j=1}^{n_i} C_{i[j]} = \sum_{i=1}^{m} \sum_{j=1}^{n_i} (n_i - j + 1) p_{i[j]}$$

  - match m "1" coefficients with largest m p_j's
  - match m "2" coefficients with next largest m p_j's, etc...
- Dispatching implementation:
  1. place jobs in SPT order
  2. assign next job on list to least loaded machine (*or* in rotation) until all jobs are scheduled
  - easily extended to *dynamic case*

Parallel Machine Models

Total Weighted Completion Time P_m / / $\Sigma w_j C_j$

- WSPT is *not* optimal for P_m | | $\Sigma w_j C_j$
  - but provides a very effective heuristic
- WSPT dispatching implementation:
  1. place jobs in WSPT order ---> $p_{[1]}/w_{[1]} < p_{[2]}/w_{[2]} < \ldots$
  2. assign next job on list to least loaded machine until all jobs are scheduled
- WSPT Worst case bound on optimality for P_m | | $\Sigma w_j C_j$

$$\frac{\sum_{j=1}^{n} w_j C_j(WSPT)}{\sum_{j=1}^{n} w_j C_j(OPT)} < \frac{1}{2}\left(1+\sqrt{2}\right) = 1.207$$

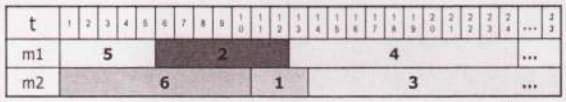
Parallel Machine Models

Total Weighted Completion Time P_m / / $\Sigma w_j C_j$

- WSPT Example:
  - m=2
  - *Data:*

job j	1	2	3	4	5	6
p_j	3	7	20	12	5	10
w_j	1	3	4	4	3	5
p_j / w_j	3	2.33	5	3	1.67	2

  - WSPT ordering: 5-6-2-1-4-3



Parallel Machine Models

Total Completion Time $P_m \mid\mid \Sigma C_j$

- Precedence Constraints: $P_m \mid prec \mid \Sigma C_j$
  - *Critical Path (CP) rule* provides optimal schedule for $P_m \mid p_j=1, \text{outtree} \mid \Sigma C_j$
  - CP rule not optimal for intrees...
  - *CP rule:* highest priority to job in front of longest "string" of jobs
- Machine Restrictions: $P_m \mid p_j=1, M_j \mid \Sigma C_j$
  - *Least Flexible Job (LFJ) rule* provides optimal schedule if M_j nested
  - Among the available jobs when a machine becomes free, pick the job that can be processed on the smallest number of machines
  - *LFJ* provides heuristic if M_j not nested

Parallel Machine Models

Total Completion Time, Unrelated Machines $R_m \mid\mid \Sigma C_j$

- R_m: processing times of jobs are machine dependent with no systematic pattern
- Integer Programming formulation:
  - let $x_{ikj}=1$, if job j in position k on machine i
    $=0$, otherwise

$$\min \sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{n}(n+1-k)p_{ij}x_{ikj}$$

$$s.t. \quad \sum_{i=1}^{m}\sum_{k=1}^{n}x_{ikj}=1 \;,\; j=1,\ldots,n$$ ← each job put in 1 position

$$\sum_{j=1}^{n}x_{ikj}\le 1 \;,\; i=1,\ldots,m \;\; k=1,\ldots,n$$ ← each position has 0 or 1 jobs assigned to it

$$x_{ikj}\in\{0,1\} \;,\; \forall i,k,j$$

Parallel Machine Models

Total Completion Time, Unrelated Machines $R_m \mid\mid \Sigma C_j$

- Integer Programming formulation:
  - a *weighted bipartite matching problem*
  - coefficient structure in the constraints guarantees a 0-1 solution
    thus, x_{ikj} integer restriction replaced by nonnegativity constraint
    thus, solved as a linear program
  - inserted idle time possible in the solution

Parallel Machine Models

Due Date-related Issues

- A heuristic approach for $P_m \mid\mid \Sigma T_j$

  Reference: Dogramaci & Surkis (1979), "Evaluation of a Heuristic for Scheduling Independent Jobs on Parallel Identical Processors", Management Science, Vol. 25, #12

  1. form priority list of n jobs
  2. assign next job from list to least loaded machine
  3. solve the $1 \mid\mid \Sigma T_j$ problem
  4. repeat steps (1), (2), and (3) for priority rules SPT, EDD, MS and choose the best schedule generated

Job Shop Scheduling
(with "Flow Shop" special case)

ISE 419

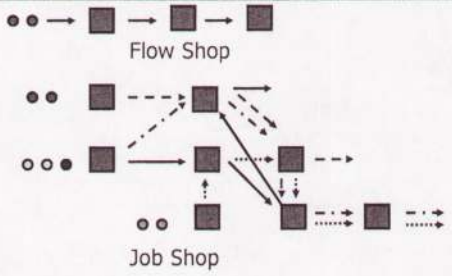
1

Job Shop Scheduling

- Have m machines and n jobs
- Each job visits some or all of the machines
  - each job follows a predetermined route
  - routes are not necessarily the same for each job
  - machine can be visited once or more than once (recirculation)
- Customer order of small batches
  - Wafer fabrication in semiconductor industry
  - Hospital
- Very difficult to solve

2

Machine Configurations



3

Machine characteristics

- **Open shop** O_m
  - m machines
  - each job has to be processed on each of the m machines
  - scheduler determines the route for each job
- **Job shop** J_m
  - m machines
  - each job has its own route
  - job may visit a machine more then once (recirculation)

  $Fm \mid recrc \mid ...$

4

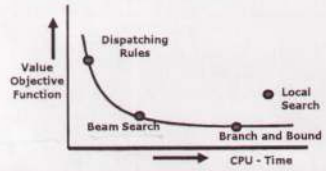
General Solution Techniques

- Mathematical programming
  - Linear, non-linear, and integer programming
- Enumerative methods
  - Branch-and-bound
  - Beam search
- Local search
  - Simulated annealing/genetic algorithms/tabu search/neural networks.
- Heuristics
  - dispatching rules
  - composite dispatching rules
  - beam-search

5

Scheduling algorithms

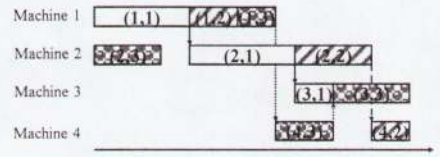
- Decomposition Techniques
  - Temporal decomposition (rolling horizon approach)
  - Machine decomposition (Shifting Bottleneck)
- Hybrid Methods
  - combined usage of scheduling methods



6

Job Shop Example

- Constraints
  - Job follows a specific route
  - One job at a time on each machine



7

Flow Shop Scheduling

- Machines may be numbered so that
  operation i preceding operation j *implies*
  machine # for i < machine # for j

- Assembly lines or staged fabrication processes
- Assumptions:
  - r_j=0
  - p_{kj} = processing time of operation for job *j* performed on machine *k* (setup times included)
  - machines always available
  - operations nonpreemptable

input output

8

Machine characteristics

Flow shop F_m
- *m* machines in series
- all jobs have the same routing
- each job has to be processed on each one of the *m* machines (permutation)
  first in first out (FIFO)
  $F_m \mid prmu \mid \ldots$

Flexible flow shop FF_s
- *s* stages in series with a number of machines in parallel
- at each stage job *j* requires only one machine
- FIFO discipline is usually between stages

9

Flow Shop Scheduling

Unlimited Intermediate Storage ***Fm \| \| C_{max}***

- Fm | | C_{max} : *m* machine flow shop with sufficient space between machines to prohibit blocking of jobs
- Permutation schedule: same job sequence used on each machine
  - Number of arbitrary schedules, $S=(n!)^m$ ... n=50, m=5, $S=2.6(10)^{322}$
  - *Sufficient* to consider only schedules with same sequence on machines 1 and 2 to optimize *any* regular measure of performance

m_1	---	(1,i)	(1,j)	---			
m_2				---	(2,j)	---	(2,i)

  - if (1,i) and (1,j) interchanged
    - no net delay on machine 1
    - (2,i) not delayed
    - (2,j) completion time *may be improved*

10

Flow Shop Scheduling

Unlimited Intermediate Storage ***Fm \| \| C_{max}***

- Permutation schedule: same job sequence used on each machine
  - *Sufficient* to consider only schedules with same sequence on machines *m-1* and *m* to optimize makespan

m_{m-1}	---	(m-1,i)	---	(m-1,j)	---		
m_m				---		(m,j)	(m,i)

  - if (m-1,i) and (m-1,j) interchanged
    - no net delay on machine m-1
    - (m,i) and (m,j) completion time *may both be improved*
  - Permutation schedules optimal for P2 | | C_{max} and P3 | | C_{max}

11

Flow Shop Scheduling

Unlimited Intermediate Storage ***Fm \| \| C_{max}***

Johnson's Rule:
SPT(1) - LPT(2) Scheduling optimal for F2 | | C_{max}

- Set up a Column Array with the number of slots equal to the total number of jobs
- Select the smallest processing time at either machine. If this is at work center 1, put this job as near to the beginning of the schedule at possible
- If the smallest time is at work center 2, put this job as near the end of the schedule as possible
- Remove the scheduled job from the list
- Repeat steps until all jobs have been scheduled

12

Flow Shop Scheduling

Unlimited Intermediate Storage ***Fm | | C_{max}***

Johnson's Rule:
SPT(1) - LPT(2) Scheduling optimal for F2 | | C_{max}

- *Motivation:*

$$\sum_{i=1}^{n} p_{1,[i]} + p_{2,[n]} \le C_{max}$$

$$p_{1,[1]} + \sum_{i=1}^{n} p_{2,[i]} \le C_{max}$$

- Summations not affected by sequencing
- Selection of $p_{1,[1]}$ and $p_{2,[n]}$ *critical*

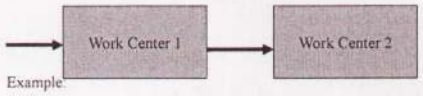
13

Flow Shop Scheduling

Unlimited Intermediate Storage ***Fm | | C_{max}***

Scheduling Through Two Work Centers

Assume all jobs go through the process in the same order.



Example:

Job	Processing Time # 1	Processing Time # 2
A	6	8
B	11	6
C	7	3
D	9	7
E	5	10

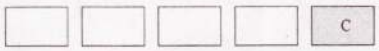
14

Flow Shop Scheduling

Unlimited Intermediate Storage ***Fm | | C_{max}***

Step 1

Job	Processing Time # 1	Processing Time # 2
A	6	8
B	11	6
C	7	3
D	9	7
E	5	10



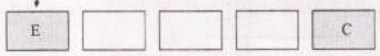
15

Flow Shop Scheduling

Unlimited Intermediate Storage ***Fm | | C_{max}***

Step 2

Job	Processing Time # 1	Processing Time # 2
A	6	8
B	11	6
~~C~~	~~7~~	~~3~~
D	9	7
E	5	10



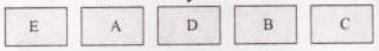
16

Flow Shop Scheduling

Unlimited Intermediate Storage ***Fm | | C_{max}***

Step 3-5

Job	Processing Time # 1	Processing Time # 2
A	6	8
B	11	6
~~C~~	~~7~~	~~3~~
D	9	7
E	5	10

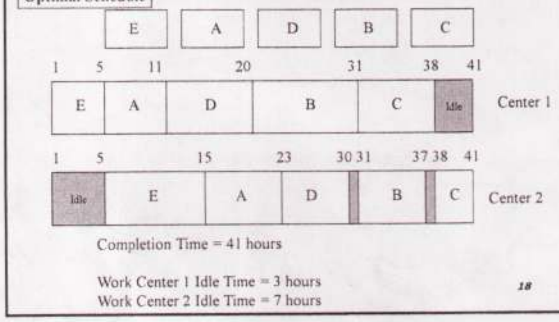


17

Flow Shop Scheduling

Unlimited Intermediate Storage ***Fm | | C_{max}***

Optimal Schedule



Completion Time = 41 hours

Work Center 1 Idle Time = 3 hours
Work Center 2 Idle Time = 7 hours

18

should be i.

Flow Shop Scheduling
Unlimited Intermediate Storage ***$Fm \mid \mid C_{max}$***

- <u>Calculating permutation schedule job completion times</u>
  - tabular format
  - let rows be ordered set of machines
  - let columns be the jobs ordered as in the solution sequence
  - <u>first row:</u> add job processing time to completion time of preceding job on first machine (no inserted idle time possible)
  - <u>first column:</u> add processing time of first job on a machine to the completion time of the first job on the previous machine
  - <u>all other entries:</u> add the processing time of job *i* on machine *k* to the maximum of
    - completion time of job *i* on machine *k-1*
    - completion time of previous job on machine *k*

19

Flow Shop Scheduling
Unlimited Intermediate Storage ***$Fm \mid \mid C_{max}$***

- <u>First row:</u> $C_{1,[i]} = \sum_{j=1}^{i} p_{1,[j]}$ i=1,...,n
- <u>First column:</u> $C_{k,[1]} = \sum_{h=1}^{k} p_{h,[1]}$ k=1,...,m

Job	$p_{1,i}$	$p_{2,i}$
A	6	8
B	11	6
C	7	3
D	9	7
E	5	10

Opt Seq	E	A	D	B	C
m_1	5	5+6=11	11+9=20	20+11=31	31+7=38
m_2	5+10=15				

- <u>Other entries:</u> $C_{k,[i]} = \max\{C_{k-1,[i]}, C_{k,[i-1]}\} + p_{k,[i]}$

Opt Seq	E	A	D	B	C
m_1	5	5+6=11	11+9=20	20+11=31	31+7=38
m_2	5+10=15	15+8=23	23+7=30	31+6=37	38+3=41

20

3 machine same sequence. — (3 cases.)
Johnson Rule extension min (Cmax).

Flow Shop Scheduling
Unlimited Intermediate Storage ***$F3 \mid dom\ p_{2,i} \mid C_{max}$***

- $F3 \mid \mid C_{max}$ strongly NP-hard
- $F3 \mid dom\ p_{2,i} \mid C_{max}$ optimized with Johnson Rule extension
- Machine 2 is *dominated* if

$$\min_i\{t_{1,i}\} \geq \max_i\{t_{2,i}\} \quad \text{or} \quad \min_i\{t_{3,i}\} \geq \max_i\{t_{2,i}\}$$

- case 1: If machine 2 is dominated, use Johnson' Rule with $\min\{t_{1,i}+t_{2,j}, t_{2,i}+t_{3,j}\}$ (m1 - m2) ⟺ solution (m2 - m3)
- case 2: If machine 2 is not dominated but m_1/m_2 problem gives same results as m_2/m_3 problem, *sequence is optimal*
- case 3: If machine 2 is <u>not</u> dominated and m_1/m_2 problem <u>does not</u> gives same results as m_2/m_3 problem (heuristic application...)
  - <u>Apply 3 machine Johnson Rule</u>
  - <u>Perform API neighborhood search</u> (optimal sequence discovered very often)

21

Flow Shop Scheduling
Unlimited Intermediate Storage ***$F3 \mid dom\ p_{2,i} \mid C_{max}$***

- <u>3-Machine Flow Shop Example</u>

job i	1	2	3	4	5	
$p_{1,i}$	12	10	14	9	16	
$p_{2,i}$	2	8	6	3	6	Dominated
$p_{3,i}$	13	8	20	11	15	
$p_{1,i}+p_{2,i}$	14	18	20	12	22	
$p_{2,i}+p_{3,i}$	15	16	26	14	21	

- <u>Johnson Rule Schedule:</u> **S** = 4 - 1 - 3 - 5 - 2

22

3 machine same sequence min (Cmax).
Extension of 3-machine Johnson's rule.

Flow Shop Scheduling permutation..
Unlimited Intermediate Storage ***$Fm \mid prmu \mid C_{max}$***

- Campbell, Dudek, Smith (CDS) heuristic [1970]
  - Extension of 3-machine Johnson's Rule logic
  - At stage *j*, calculate

$$p'_{1,i} = \sum_{k=1}^{j} p_{k,i} \quad \text{and} \quad p'_{2,i} = \sum_{k=1}^{j} p_{m-k+1,i}$$

and apply Johnson's Rule to the "2-machine" problem
  - After *m-1* stages, pick the sequence generated at some stage with the best makespan

23

对 stage j. ($P_{1,j}$ Σ first j stages 的 ΣP; $P_{2,j}$ Σ last j stages 用Σ)
设定 $P'_{1,j}$ 和 $P'_{2,j}$ 根据 Johnson's Rule 排.

Flow Shop Scheduling
Unlimited Intermediate Storage ***$Fm \mid prmu \mid C_{max}$***

- <u>CDS heuristic example:</u>

job i	1	2	3	4
$p_{1,i}$	10	12	6	5
$p_{2,i}$	13	8	9	10
$p_{3,i}$	7	6	8	9
$p_{4,i}$	6	3	7	12

stage 1	p'(1,i)	10	12	6	5	⇨ S=4-3-1-2
	p'(2,i)	6	3	7	12	
stage 2	p'(1,i)	23	20	15	15	⇨ S=4-3-1-2
	p'(2,i)	13	9	15	21	
stage 3	p'(1,i)	30	26	23	24	⇨ S=3-4-1-2
	p'(2,i)	26	17	24	31	

24

排序几个比中最优.

Flow Shop Scheduling

Unlimited Intermediate Storage ***Fm / prmu / C_{max}***

❖ Schedule evaluations

S =	4	3	1	2	
m_1	5	11	21	33	
m_2	15	24	37	45	
m_3	24	32	44	51	
m_4	36	43	50	54	← Best

S =	3	4	1	2
m_1	6	11	21	33
m_2	15	25	38	46
m_3	23	34	45	52
m_4	30	46	52	55

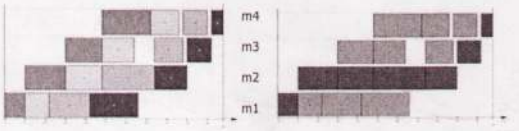
25

Flow Shop Scheduling

Unlimited Intermediate Storage ***Fm / prmu / C_{max}***

❖ Gantt chart of solution S=4-3-1-2

– Chart on right highlights Critical path



26

Flow Shop Scheduling

Unlimited Intermediate Storage ***Fm / prmu / C_{max}***

❖ Mixed-integer programming formulation

❖ Definitions:

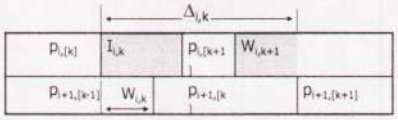
– $x_{j,k}$ = 1, if job j in position k of sequence

– $I_{i,k}$ = idle time on machine i between jobs in positions k and $k+1$

– $W_{j,k}$ = waiting time of job in position k between machines i and $i+1$

– $\Delta_{i,k} = I_{i,k} + p_{i,[k+1]} + W_{i,k+1} = W_{i,k} + p_{i+1,[k]} + I_{i+1,k}$

= difference in starting time of position $k+1$ job on machine $i+1$ and the completion time of the position k job on machine i



27

Flow Shop Scheduling

Unlimited Intermediate Storage ***Fm / prmu / C_{max}***

❖ Mixed-integer programming formulation

– *Minimize makespan* ⟺ *minimize* idle time on last *machine*

$= \sum_{j=1}^{n} p_{j,[1]} + \sum_{i=1}^{m-1} I_{m,j}$ = time for job [1] to get to machine m + sum of idle intervals on machine m

$\min \sum_{i=1}^{m-1} \sum_{j=1}^{n} x_{j,1} p_{i,j} + \sum_{j=1}^{n} I_{m,j}$

s.t.

$\sum_{j=1}^{n} x_{j,k} = 1 \quad k=1,\ldots,n$

$\sum_{k=1}^{n} x_{j,k} = 1 \quad j=1,\ldots,n$

assignment problem constraints

$I_{i,k} + \sum_{j=1}^{n} x_{j,k} p_{i,j} + W_{i,k} - [I_{i,k+1} + \sum_{j=1}^{n} x_{j,k+1} p_{i+1,j} + W_{i,k+1}] = 0 \quad k=1,\ldots,n-1 \quad i=1,\ldots,m-1$

$W_{1,k} = 0 \quad i=1,\ldots,m-1$

$I_{m,k} = 0 \quad k=1,\ldots,n-1$

$I_{i,k} + p_{i,[k+1]} + W_{i,k+1} = W_{i,k} + p_{i+1,[k]} + I_{i+1,k}$

28

Flow Shop Scheduling

Special Features Flow Shops

❖ Machines have different speeds in *proportionate permutation flow shop*... ($p_{mj} = p_j$ all m)

– machine with min v_k is *bottleneck* (v_k = speed of machine k)

– if first machine is bottleneck ---> LPT minimizes C_{max}

– if last machine is bottleneck ---> SPT minimizes C_{max}

❖ Flexible Flow Shop

– c stages

– each stage i has m_i identical processors

- makespan heuristic: LPT (LRPT , if preemption)
- sum of C_i heuristic: SPT , if $m_i \geq m_{i-1}$ i=2,...,c
  - jobs will not be blocked at next stage
  - SPT minimizes start time on m_1 and total processing time

29

Flow Shop Scheduling

Limited Intermediate Storage ***Fm / (nwt) / C_{max}***

job j	$p_{1,j}$	$p_{2,j}$		$p_{3,j}$	.....	$p_{m,j}$	
job k		$p_{1,k}$	$p_{2,k}$	$I_{3,k}$	$p_{3,k}$	.....	$p_{m,k}$

❖ "No-wait" flow shop

– jobs are not blocked once they start through the shop

– jobs "pulled" to next machine

– push all job k times to the right

– group all idle periods with respect to job j at the beginning

– more restrictive --> $C_{max}(nwt) \geq C_{max}(blocking)$

$d_{j,k}$ = total delay to job k with respect to job j

$= \sum_{i=1}^{m} I_{i,k}$

30

MIP.

the total idle time on the last machine.

Let $P_{i(k)} = \sum_{j=1}^{n} x_{j,k} p_{i,j}$.

$\min \left(\sum_{i=1}^{m-1} \sum_{j=1}^{n} x_{j,1} p_{i,j} + \sum_{j=1}^{n} I_{m,j} \right)$ (let k=1).

(为 limited 作准备)

$I_{1,k=0}$

NO-WAIT Flow Shop min ((max) no-wait.

$p_{ij} = \frac{p_j}{v_i}$.

紫的为 m_j → for job

m_i → for stage, part of job

把所有 idle 时间推到前面.

obviously

Start time is $(C_{i,j-1}) + (d_{j+1,j})$.

job (j-1) 第一步结束加上 job j 因 (j-1) 而所有 idle.

Flow Shop Scheduling

Limited Intermediate Storage — *Fm | nwt | C_{max}*

❖ Total delay to job k with respect to job j calculation

job j	$p_{1,j}$	$p_{2,j}$		$p_{k,j}$	.....	$p_{m,j}$	
job k		$p_{1,k}$	$p_{2,k}$	$I_{3,k}$	$p_{3,k}$	.....	$p_{m,k}$

$$I_{1,k} = p_{1,j}$$

$$I_{2,k} = \max\{0, p_{1,j} + p_{2,j} - p_{1,k} - I_{1,k}\}$$

$$I_{3,k} = \max\{0, p_{1,j} + p_{2,j} + p_{3,j} - p_{1,k} - p_{2,k} - I_{1,k} - I_{2,k}\}$$

in general,

$$I_{i,k} = \max\left\{0, \sum_{h=1}^{i} p_{h,j} - \sum_{h=1}^{i-1}[p_{h,k} + I_{h,k}]\right\}$$

$$d_{j,k} = \max_{1 \le i \le m}\left\{\sum_{h=1}^{i} p_{h,j} - \sum_{h=1}^{i-1} p_{h,k}\right\}$$

minimized using TSP

$$C_{max} = \sum_{i=1}^{n-1} d_{[i],[i+1]} + \sum_{i=1}^{m} p_{i,[n]}$$

constant

31

Flow Shop Scheduling

Limited Intermediate Storage — *Fm | nwt | C_{max}*

❖ TSP distance matrix, D

	⋮			$\sum_{i=1}^{m} p_{i,1}$
...	$d_{j,k}$	...		$\sum_{i=1}^{m} p_{i,2}$
	⋮			⋮
				$\sum_{i=1}^{m} p_{i,n}$
0	0	...	0	.....

32

Flow Shop Scheduling

Limited Intermediate Storage — *Fm | nwt | C_{max}*

❖ No-wait example

$p_{i,j}$	j1	j2	j3	j4
m1	3	11	7	10
m2	4	1	9	12
m3	10	5	13	2

j	k	$d_{j,k} = \max_{1 \le i \le m}\left\{\sum_{h=1}^{i} p_{h,j} - \sum_{h=1}^{i-1} p_{h,k}\right\}$	max=
1	2	max{ 3, 7-11, 17-12 }	5
1	3	max{ 3, 7-7, 17-16 }	3
1	4	max{ 3, 7-10,17-22 }	3
2	1	max{ 11, 12-3, 17-7 }	11
2	3	max{ 11, 12-7,17-16 }	11
2	4	max{ 11, 12-10, 17-22 }	11
3	1	max{ 7, 16-3, 29-7 }	22
3	2	max{ 7, 16-11, 29-12 }	17
3	4	max{ 7, 16-10, 29-22 }	7
4	1	max{ 10, 22-3, 24-7 }	19
4	2	max{ 10, 22-11, 24-12 }	12
4	3	max{ 10, 22-7, 24-16 }	15

33

Flow Shop Scheduling

Limited Intermediate Storage — *Fm | nwt | C_{max}*

❖ Distance matrix for example

Closest unvisited city heuristic for TSP
- try all jobs in starting position

D	j1	j2	j3	j4	Dum
j1	.....	5	3	3	17
j2	11	.....	11	11	17
j3	22	17	.....	7	29
j4	19	12	15	.....	24
Dum	0	0	0	0	.....

Sequence	distance
1-3-4-2	39*
1-4-2-3	55
2-1-3-4	45
2-1-4-3	58
2-3-4-1	54
2-4-3-1	65
3-4-2-1	47
4-2-3-1	62
4-2-1-3	55

34

No-wait Flow shop min (ΣC_j)

Flow Shop Scheduling

Limited Intermediate Storage — *Fm | nwt | ΣC_j*

Reference: C. Rajendran and D. Chaudhuri (1990), "Heuristic Algorithms for Continuous Flow Shop Problem", Naval Research Logistics, 37 , 695-705.

❖ Objective is to minimize the sum of the completion times of the n jobs on machine m

$$\sum_{j=2}^{n}(n+1-i)\,d_{[i-1],[i]} + \sum_{i=1}^{n}\sum_{j=1}^{m} p_{j,i}$$

constant

$q(\sigma,1)$ = completion time of σ on machine 1

$LB(\sigma i, j)$ = lower bound on the completion time of job i on machine j

$$= q(\sigma,1) + \sum_{p=1}^{j} p_{p,i}$$

$$S_i = \sum_{j=1}^{m} LB(\sigma i, j) = mq(\sigma,1) + \sum_{j=1}^{m}(m-j+1)p_{j,i}$$

35

Flow Shop Scheduling

Limited Intermediate Storage — *Fm | nwt | ΣC_j*

❖ Heuristic approach based on job insertion
- σi is preferred to σk if $S_i \le S_k$
- indicates there is less "pressure" on the completions of jobs following job i vs. job k across all machines

❖ Algorithm

1. Order jobs by ascending values of $T_i = \sum_{j=1}^{m}(m-j+1)p_{j,i}$
   - ties broken by selecting job with smallest $T_i' = \sum_{j=1}^{m} p_{j,i}$
2. Place first job on list in position p of partial schedule, where $\frac{n'+1}{2} \le p \le n'+1$, n' = number of jobs in σ

36

Flow Shop Scheduling

Limited Intermediate Storage *Fm | nwt | ΣC_j*

- Algorithm (cont.)
  3. For each partial schedule generated, evaluate
     $D_p = \sum_{i=2}^{n'+1} (n'+2-i) d_{[i-1],[i]}$, for each value of p
     and choose partial schedule with minimum D_p
  4. Set σ = chosen partial schedule and n' = n'+1
  5. Delete first job from T_i ordered list
  6. If T_i empty --> stop, schedule completed
     Else, go to step (2)

Flow Shop Scheduling

Limited Intermediate Storage *Fm | nwt | ΣC_j*

- Example: m = 4, n = 5

$P_{j,i}$	j1	j2	j3	j4	j5
m1	12	20	19	14	19
m2	24	3	20	23	15
m3	12	19	3	16	17
m4	13	11	15	14	22

- form ordered list from T_i and T_i'

job i	1	2	3	4	5
T_i	157	138	157	171	177
T_i'	61	53	57	67	73

ordered list = 2-3-1-4-5
(T_i' used to order 3-1)

Flow Shop Scheduling

Limited Intermediate Storage *Fm | nwt | ΣC_j*

- Distance matrix: $d_{j,k} = \max_{1 \le i \le m} \left\{ \sum_{h=1}^{i} p_{h,j} - \sum_{h=1}^{i-1} p_{h,k} \right\}$

$d_{j,k}$	1	2	3	4	5
1	---	25	19	22	17
2	20	---	20	20	20
3	27	19	---	25	20
4	25	30	25	---	19
5	25	31	31	20	---

Flow Shop Scheduling

Limited Intermediate Storage *Fm | nwt | ΣC_j*

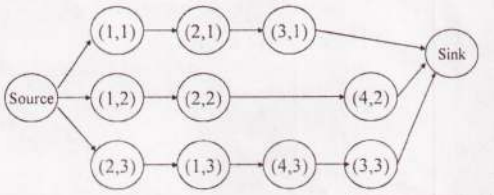
- Algorithm $D_p = \sum_{i=2}^{n'+1} (n'+2-i) d_{[i-1],[i]}$, for each value of p

σ	n'	job from list	candidate σ	D_p	chosen σ
Φ	0	2	2	---	2
2	1	3	3-2, 2-3	19, 20	3-2
3-2	2	1	3-1-2, 3-2-1	79, 58	3-2-1
3-2-1	3	4	3-4-2-1, 3-2-4-1, 3-2-1-4	155, 122, 119	3-2-1-4
3-2-1-4	4	5	3-2-5-1-4, 3-2-1-5-4, 3-2-1-4-5	208, 190, 199	3-2-1-5-4

- Sum of completion times $= 190 + \sum_{i=1}^{5} \sum_{j=1}^{4} p_{j,i} = 501$
- ***This is the optimal schedule***

Graph Representation of Job Shop Problem

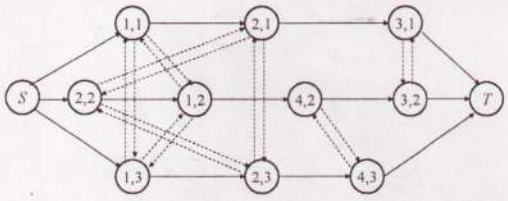
Each job follows a specific route through the job shop ...



Conjunctive arcs

Graph Representation of Job Shop Problem

Example of a job shop problem: 4 machines and 3 jobs



Disjunctive Graph

Graph Representation of Job Shop Problem

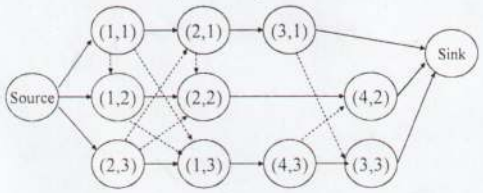
- (i, j) processing of job j on machine i
- p_{ij} processing time of job j on machine i
- $G = (N, A \cup B)$
- $(i, j) \in N$ all the operations that must be performed on the n jobs
- A **conjunctive** (solid) arcs represent the precedence relationships between the processing operations of a single job
- B **disjunctive** (broken) arcs connect two operations which belong to two different jobs, that are to be processed on the same machine, they go in opposite directions

- Disjunctive arcs form a clique for each machine.
- Clique is a maximal subgraph in which all pairs of nodes are connected with each other.
- Operations in the same clique have to be done on the same machine.

43

Solving the Makespan Problem

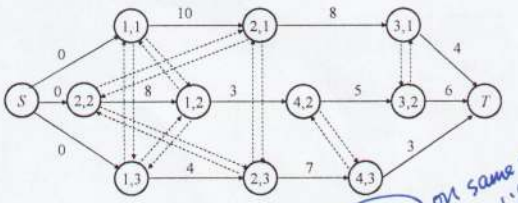
- Select one of each pair of disjunctive arcs



- The longest path in this graph $G(D)$ determines the makespan

44

Solving the Makespan Problem



- Minimize makespan: find a selection of disjunctive arcs that minimizes the length of the longest path (the critical path).

45

Disjunctive Programming

Minimize $C_{\max}$

Subject to

$$y_{kj} - y_{ij} \geq p_{ij} \quad \text{for all } (i,j) \rightarrow (k,j)$$

$$C_{\max} - y_{ij} \geq p_{ij} \quad \text{for all } (i,j)$$

$$\begin{array}{|l} y_{ij} - y_{il} \geq p_{il} \\ \text{or } y_{il} - y_{ij} \geq p_{ij} \end{array} \quad \text{for all } (i,j) \text{ and } (i,l)$$

$$y_{ij} \geq 0 \quad \text{for all } (i,j)$$

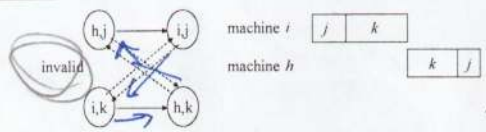
46

Graph Representation

How to construct a feasible schedule (*A Selection*)

- Select D: a subset of disjunctive arcs (one from each pair) such that the resulting directed graph $G(D)$ has no cycles.
- Graph $G(D)$ contains conjunctive arcs + D.
  - D represents a feasible schedule.
  - A cycle in the graph corresponds to a schedule that is infeasible.

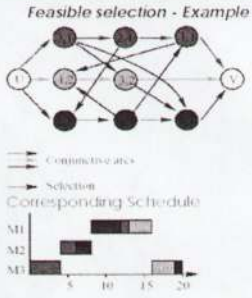
Example



47

Graph Representation

Feasible selection - Example

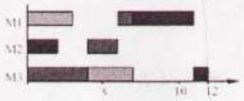


Makespan Cmax = 20

48

Graph Representation

Selection for given Schedule



Makespan Cmax 12

Corresponding selection

Conjunctive arcs — Selection

49

Job Shop Scheduling

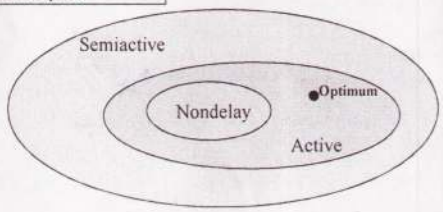
Definitions

- A schedule is **nondelay** if no machine is idled when there is an operation available
- A schedule is called **active** if no operation can be completed earlier by altering the sequence on machines and not delaying other operations
- For "regular" objectives the optimal schedule is always active but not necessarily nondelay

50

Job Shop Scheduling

Schedule Space



All Schedules

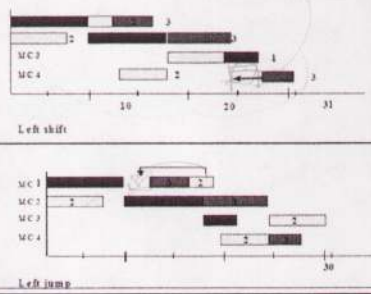
51

Job Shop Scheduling

Active Schedules:

Two types of modification of a feasible schedule to obtain another feasible schedule

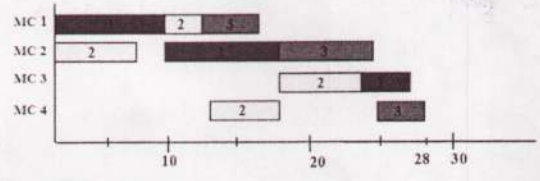
- A *left shift* move an operation left to start it earlier
- A *left jump* move an operation left into an idle slot to start it earlier



52

Job Shop Scheduling

- A feasible schedule is *active* if it cannot be modified by a left shift or a jump to complete an operation earlier without completing another operation later

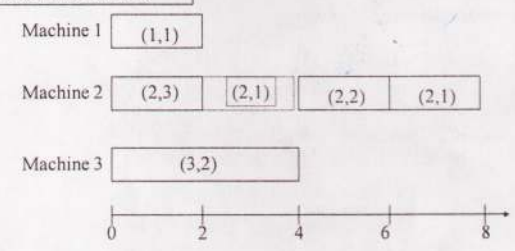


Active schedule

53

Job Shop Scheduling

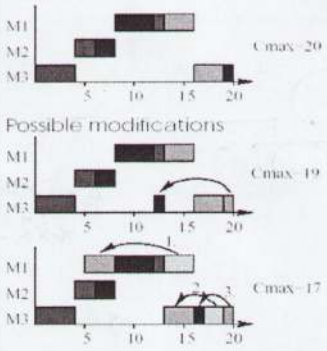
Nonactive Schedule



54

Job Shop Scheduling

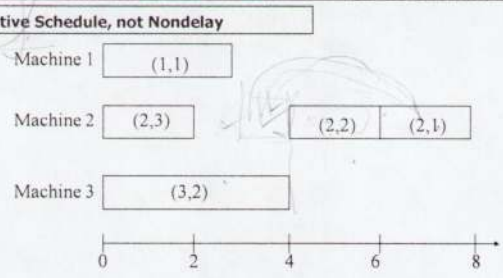
Example of a non-active schedule



55

Job Shop Scheduling

Active Schedule, not Nondelay

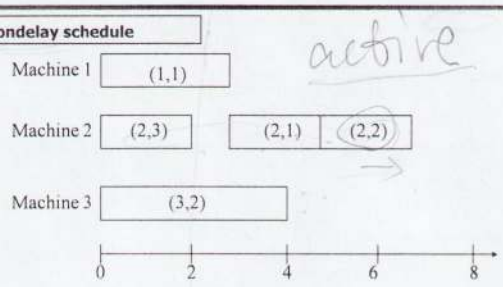


56

Jm different machine, not same sequence. min (C_{max}).

Job Shop Scheduling

Nondelay schedule



active

57

Job Shop Scheduling

Jm | | C_{max}

Branch and Bound

- Operation (i,j) with duration p_{ij}
- Minimize makespan
- Branch by generating all active schedules
- Notation
  - Let Ω denote operations whose predecessors have been scheduled
  - Let r_{ij} be the earliest possible starting time of (i,j) in Ω.

58

Job Shop Scheduling

Jm | | C_{max}

Generating Active Schedules

Step 1. (Initialize)
Let Ω contain the first operation of each job
Let $r_{ij} = 0$ for all $(i,j) \in \Omega$.

Step 2. (Machine selection)
Compute for the current partial schedule

$$t(\Omega) = \min_{(i,j)\in\Omega} \{ r_{ij} + p_{ij} \}$$

and let i^* denote the machine where minimum achieved

59

Job Shop Scheduling

Jm | | C_{max}

Step 3. (Branching)
Let Ω' denote all operations on machine i^* such that

$$\Omega' = \{ (i,j) \in \Omega \mid r_{ij} < t(\Omega) = \min_{(i,j)\in\Omega} \{ r_{ij} + p_{ij} \} \}.$$

P68 example

For each operation in Ω' consider a partial schedule with that operation next on i^*

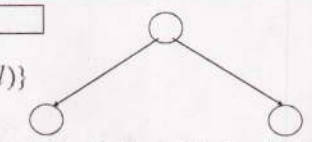
For each partial schedule, delete operation from Ω and include immediate follower in Ω. Go back to Step 2.

60

Job Shop Scheduling
Jm | | C_{max}

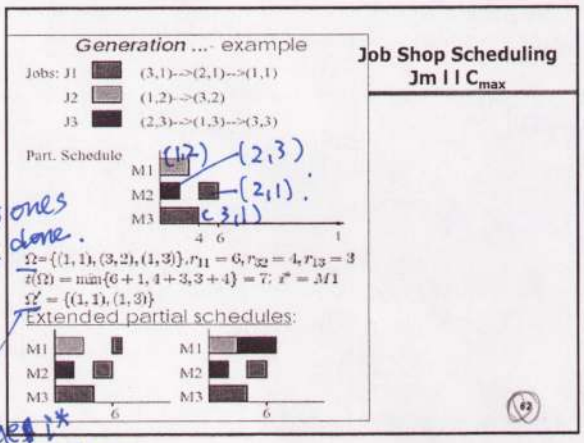
Branching Tree

$\Omega' = \{(i^*, j), (i^*, l)\}$



❖Each node in the B&B tree is characterized by a set D' of fixed disjunctions
- ➢Add disjunctions (i*,j) --> (i*,k) for all unscheduled operations (i*,k)
- ➢Add disjunctions (i*,l) --> (i*,k) for all unscheduled operations (i*,k)

61



Job Shop Scheduling
Jm | | C_{max}

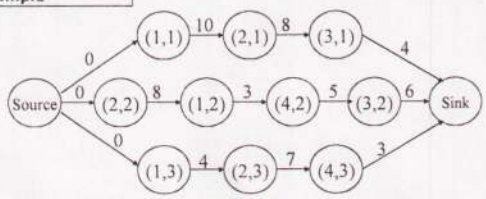
62

next ones can be done.

on chosen machine i^*
next ones can be done.

Job Shop Scheduling
Jm | | C_{max}

Example



$C_{max} \geq 22 = \max(C_j)$ (10+8+4) or (8+3+5+6).

longest path.

Critical path 已有接箭头.

63

Job Shop Scheduling
Jm | | C_{max}

◆ Level 1

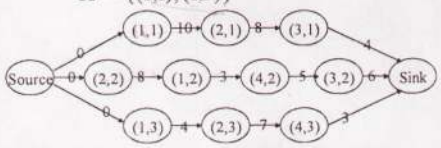
Partitioning Tree

$\Omega = \{(1,1),(2,2),(1,3)\}$

$t(\Omega) = \min\{0+10, 0+8, 0+4\} = 4$

machine $i^* = 1$

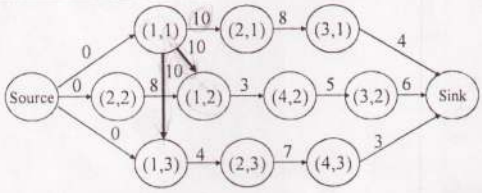
$\Omega' = \{(1,1),(1,3)\}$



64

Job Shop Scheduling
Jm | | C_{max}

Level 1: select (1,1)



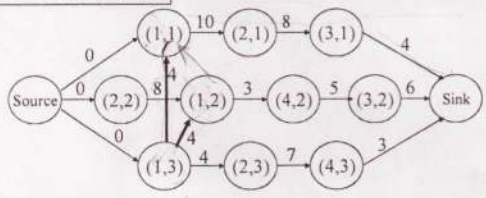
$C_{max} \geq 24$ (10+3+5+6) or (10+4+7+3)

⟶ Disjunctive Arcs

65

Job Shop Scheduling
Jm | | C_{max}

Level 1: select (1,3)



$C_{max} \geq 26$ (4+10+8+4)

⟶ Disjunctive Arcs

66

Job Shop Scheduling
Jm | | C_{max}

Branching Tree

No disjunctive arcs

(1,1) scheduled first on machine 1

LB = 24

(1,3) scheduled first on machine 1

LB = 26

67

Job Shop Scheduling
Jm | | C_{max}

- Level 2:

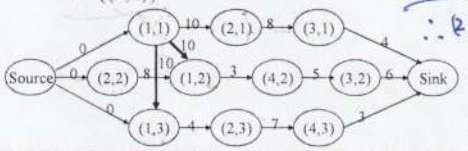
Branching Tree

$\Omega = \{(2,2), (2,1), (1,3)\}$

$t(\Omega) = \min\{0+8, 10+8, 10+4\} = 8$

$i^* = 2$

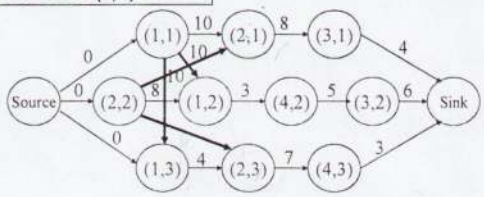
$\Omega' = \{(2,2)\}$



68

Job Shop Scheduling
Jm | | C_{max}

Level 2: Select (2,2)



69

Job Shop Scheduling
Jm | | C_{max}

Branching Tree

No disjunctive arcs

(1,1) scheduled first on machine 1

LB = 24

(1,3) scheduled first on machine 1

LB = 26

(1,1) first on M1 and (2,2) first on M2

LB = 24

70

Job Shop Scheduling
Shifting Bottleneck

The Shifting Bottleneck Heuristic and the Makespan

- A classic idea in nonlinear programming is to hold all but one variable fixed and then optimize over that variable.
- Then hold all but a different one fixed, and so on.
- If we can do the one-variable optimization in order of *decreasing importance*, there is better hope that the local optimum found will be the global one, or close to it.
- In job shop problems *fixing the value of variables* means *fixing the sequence on a given machine.*

71

Shifting Bottleneck

- Pick most loaded resource
- Find optimal one-machine schedule
- Pick next most loaded resource
- Find optimal one-machine schedule consistent with previous one-machine schedules

72

Job Shop Scheduling
Shifting Bottleneck ***Jm / / C_{max}***

Shifting Bottleneck Algorithm

Step 1. Set the initial conditions
- $M_0 = \varnothing$ set of scheduled machines.
- Graph G is the graph with all the conjunctive arcs and no disjunctive arcs.
- Set $C_{max}(\varnothing)$ equal to the longest path in graph G.

Step 2. Analysis of the machines still to be scheduled
- formulate a single machine problem with all operations subject to *release dates* and *due dates*.

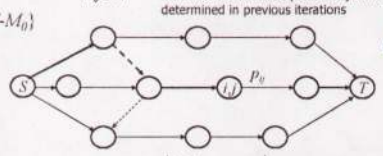
the way to find (r_{ij}) and (d_{ij}) eg P_{80} P_{87} better.

73

Job Shop Scheduling
Shifting Bottleneck

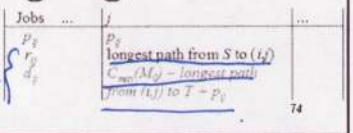
Iteration
M set of m machines
$M_0 \subset M$ machines for which sequence of jobs has already been determined in previous iterations

$i \in \{M-M_0\}$



对于未 determined 的 machine 排.

Define a single-machine problem $1 \mid r_j \mid L_{max}$ for machine i



74

Job Shop Scheduling
Shifting Bottleneck

Step 3. **Bottleneck selection**

① find the min(L_{max}) of each machine.

A machine k with the largest maximum lateness is a *bottleneck*.

$$L_{max}(k) = \max_{i \in \{M-M_0\}} (L_{max}(i))$$

② find the max one of all the min(L_{max})s that's bottleneck

1. Schedule machine k according to the sequence which minimizes the corresponding L_{max} (single-machine problem).
2. Insert all the corresponding disjunctive arcs in the graph.
3. Insert machine k in M_0.

$$C_{max}(M_0 \cup k) \geq C_{max}(M_0) + L_{max}(k)$$

75

Job Shop Scheduling
Shifting Bottleneck

Step 4. **Resequencing of all machines scheduled earlier**

Aim: to reduce the makespan

Do for each machine $l \in \{M_0 - k\}$
- delete the disjunctive arcs associated with the machine l
- formulate a single machine problem for the machine l and find the sequence that minimizes $L_{max}(l)$
- Insert the corresponding disjunctive arcs.

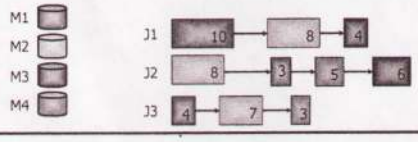
Step 5. **Go to Step 2 until $M_0 = M$**

76

Job Shop Scheduling
Shifting Bottleneck

Example

Jobs	Machines	Processing times
1	1,2,3	$p_{11}=10$, $p_{21}=8$, $p_{31}=4$
2	2,1,4,3	$p_{22}=8$, $p_{12}=3$, $p_{42}=5$, $p_{32}=6$
3	1,2,4	$p_{13}=4$, $p_{23}=7$, $p_{43}=3$



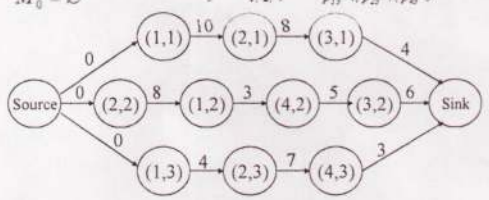
77

Job Shop Scheduling
Shifting Bottleneck

Step 1

$M_0 = \varnothing$

Jobs	Machine Sequence	Processing Times
1	1, 2, 3	$p_{11}=10$, $p_{21}=8$, $p_{31}=4$
2	2, 1, 4, 3	$p_{22}=8$, $p_{12}=3$, $p_{42}=5$, $p_{32}=6$
3	1, 2, 4	$p_{13}=4$, $p_{23}=7$, $p_{43}=3$

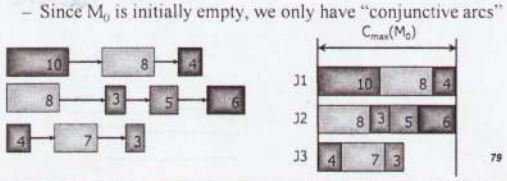


$C_{max}(M_0) = 22$ 78

Job Shop Scheduling
Shifting Bottleneck

Step 2

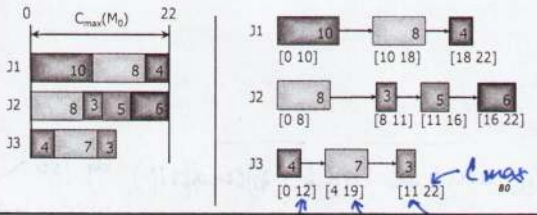
- Find release date and due date of each operation
- Remove all sequence constraints among activities in M – M_0, use CPM to find CP and min. start time, max. end time for each activity
  - Since M_0 is initially empty, we only have "conjunctive arcs"



79

Job Shop Scheduling
Shifting Bottleneck

Step 2 : Find Release & Due Dates (CPM)

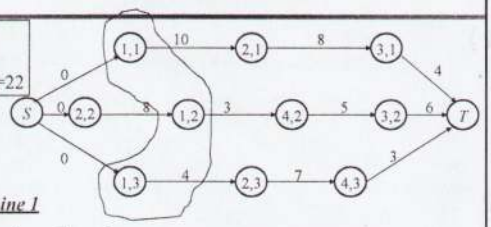


80

(22-7-3) (22-3) (4+7)

Job Shop Scheduling
Shifting Bottleneck

Step 3
$M_0 = \varnothing$
$C_{max}(\varnothing)=22$



Machine 1

Jobs	1	2	3
p_{ij}	10	3	4
r_{ij}	0	8	0
d_{ij}	10	11	12

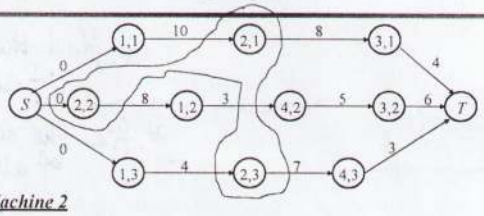
$L_{max}(1, 2, 3) = \max\{0, 2, 5\} = 5$
$L_{max}(1, 3, 2) = \max\{0, 2, 6\} = 6$
$L_{max}(2, 1, 3) = \max\{0, 11, 13\} = 13$
$L_{max}(2, 3, 1) = \max\{0, 3, 15\} = 15$
$L_{max}(3, 1, 2) = \max\{-8, 4, 6\} = 6$
$L_{max}(3, 2, 1) = \max\{-8, 0, 15\} = 15$

choose 1,2,3 L_{max}=5

81

L: lateness. $G_j - d_{ij}$.

Job Shop Scheduling
Shifting Bottleneck



Machine 2

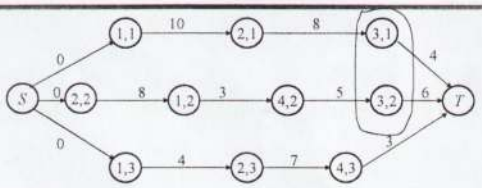
Jobs	1	2	3
p_{ij}	8	8	7
r_{ij}	10	0	4
d_{ij}	18	8	19

$L_{max}(1, 2, 3) = \max\{0, 16, 14\} = 16$
$L_{max}(1, 3, 2) = \max\{0, 6, 25\} = 25$
$L_{max}(2, 1, 3) = \max\{0, 0, 6\} = 6$
$L_{max}(2, 3, 1) = \max\{0, -4, 5\} = 5$
$L_{max}(3, 1, 2) = \max\{-8, 1, 19\} = 19$
$L_{max}(3, 2, 1) = \max\{-8, 11, 9\} = 11$

2,3,1 L_{max}=5

82

Job Shop Scheduling
Shifting Bottleneck



Machine 3

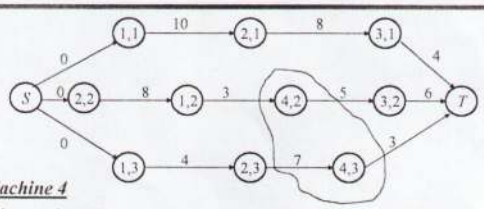
Jobs	1	2
p_{ij}	4	6
r_{ij}	18	16
d_{ij}	22	22

$L_{max}(1, 2) = \max\{0, 6\} = 6$
$L_{max}(2, 1) = \max\{0, 4\} = \underline{4}$

L_{max}=4

83

Job Shop Scheduling
Shifting Bottleneck



Machine 4

Jobs	2	3
p_{ij}	5	5
r_{ij}	11	11
d_{ij}	16	22

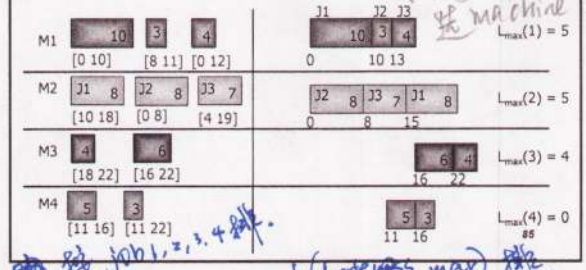
$L_{max}(2, 3) = \max\{0, -3\} = \mathbf{0}$
$L_{max}(3, 2) = \max\{-8, 3\} = 3$

L_{max}=0

84

Job Shop Scheduling
Shifting Bottleneck

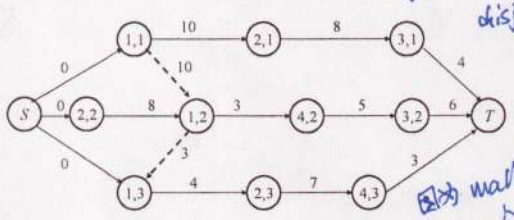
- Machines 1 and 2 are the bottlenecks
  - arbitrarily pick machine 1...



85

Job Shop Scheduling
Shifting Bottleneck

Step 2 (Iteration 2) $M_0 = \{1\}$



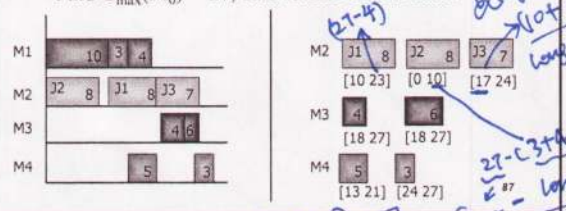
$C_{max}(\{1\}) = C_{max}(\varnothing) + L_{max}(1) = 22 + 5 = 27$

86

Job Shop Scheduling
Shifting Bottleneck

Step 2 (Iteration 2)

- CPM
  - Find $C_{max}(M_0) = 27$, find release & due dates

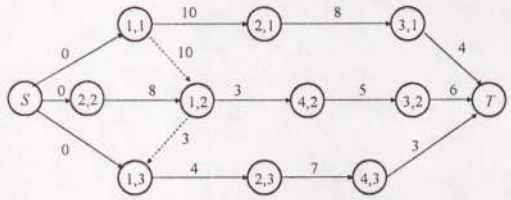


87

Job Shop Scheduling
Shifting Bottleneck

$M_0 = \{1\}$, $C_{max}(\{1\})=27$

Selecting a Machine



Machine 2

Jobs	1	2	3
p_{ij}	8	8	7
r_{ij}	10	0	17
d_{ij}	23	10	24

2,1,3 L_{max}=1

88

Job Shop Scheduling
Shifting Bottleneck

Machine 3

Jobs	1	2
p_{ij}	4	6
r_{ij}	18	18
d_{ij}	27	27

1,2 or 2,1 L_{max}=1

Machine 4 L_{max}=0

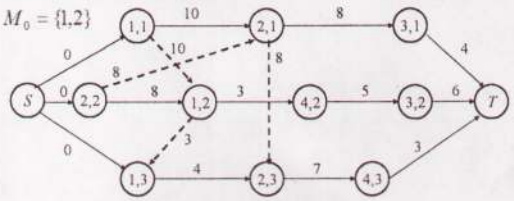
Machines 2 and 3 are bottlenecks.

Machine 2 is chosen as a bottleneck!

89

Job Shop Scheduling
Shifting Bottleneck

$M_0 = \{1,2\}$



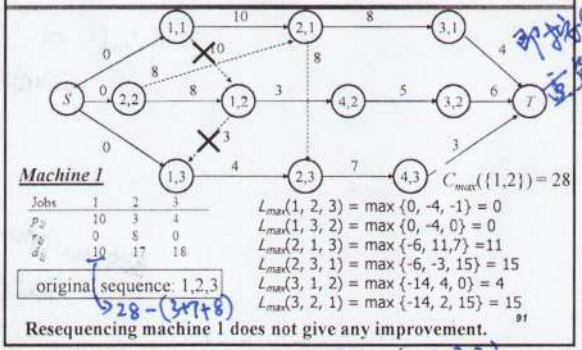
$C_{max}(\{1,2\}) = C_{max}(\{1\}) + L_{max}(2) = 27 + 1 = 28$

Can we decrease $C_{max}(\{1,2\})$?

Will resequencing machine 1 give any improvement ?

90

Job Shop Scheduling
Shifting Bottleneck



Machine 1

Jobs	1	2	3
p_{ij}	10	3	4
r_{ij}	0	8	0
d_{ij}	10	17	18

original sequence: 1,2,3

$L_{max}(1, 2, 3) = \max\{0, -4, -1\} = 0$
$L_{max}(1, 3, 2) = \max\{0, -4, 0\} = 0$
$L_{max}(2, 1, 3) = \max\{-6, 11, 7\} = 11$
$L_{max}(2, 3, 1) = \max\{-6, -3, 15\} = 15$
$L_{max}(3, 1, 2) = \max\{-14, 4, 0\} = 4$
$L_{max}(3, 2, 1) = \max\{-14, 2, 15\} = 15$

$C_{max}(\{1,2\}) = 28$

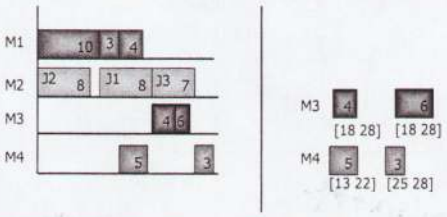
Resequencing machine 1 does not give any improvement.

91

Job Shop Scheduling
Shifting Bottleneck

Step 2 (Iteration 3)

- CPM
  - Find $C_{max}(M_0) = 28$, find release & due dates



M3 [18 28] [18 28]
M4 [13 22] [25 28]

92

(1,2,3) and (1,3,2) are still have min(Lmax).

Job Shop Scheduling
Shifting Bottleneck

Step 3
$M_0 = \{1,2\}$
$C_{max}(\{1,2\}) = 28$

Machine 3

Jobs	1	2
p_{ij}	4	6
r_{ij}	18	18
d_{ij}	28	28

1,2 or 2,1 $L_{max} = 0$

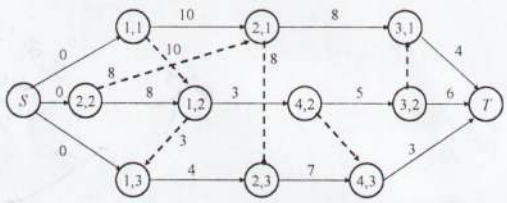
Machine 4 $L_{max} = 0$

No machine is a bottleneck!

93

Job Shop Scheduling
Shifting Bottleneck

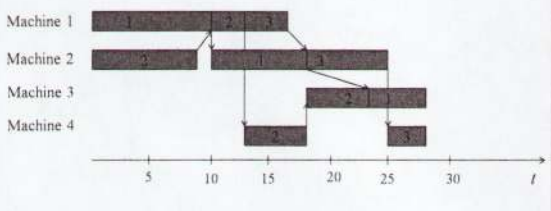
Solution Graph



94

Job Shop Scheduling
Shifting Bottleneck

Solution Gantt Chart

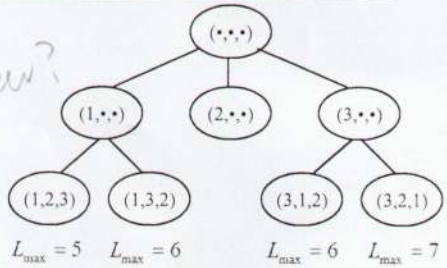


95

Job Shop Scheduling
Shifting Bottleneck *Jm | | C_{max}*

Solving the Maximum Lateness Problem

review?

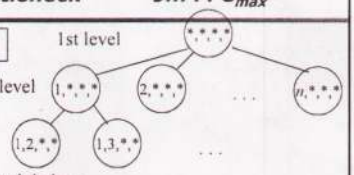


$L_{max} = 5$ $L_{max} = 6$ $L_{max} = 6$ $L_{max} = 7$

96

Job Shop Scheduling
Shifting Bottleneck ***Jm / / C_{max}***

Branch and Bound



Branching rule:

k-1 level, $j_1, \ldots, j_{k-1}$ are scheduled, $\cdots$

j_k needs to be considered if no job still to be scheduled can not be processed before the release time of j_k

that is:

$$r_{j_k} < \min_{l \in J} (\max(t, r_l) + p_l)$$

J set of jobs not yet scheduled

t is time when j_{k-1} is completed

97

Job Shop Scheduling
Shifting Bottleneck ***Jm / / C_{max}***

Relaxing the Problem

- The problem $1 \mid r_j, prmp \mid L_{max}$

  is a relaxation to the problem $1 \mid r_j \mid L_{max}$

- Not allowing preemption is a constraint in the original problem but not the relaxed problem
- We know how to solve the relaxed problem (preemptive EDD rule)

98

Job Shop Scheduling
Shifting Bottleneck ***Jm / / C_{max}***

Lower bound:

❖ Preemptive earliest due date (EDD) rule is optimal for

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

❖ A preemptive schedule will have a maximum lateness not greater than a non-preemptive schedule.

❖ If a preemptive EDD rule gives a non-preemptive schedule then all nodes with a larger lower bound can be disregarded.

99

Job Shop Scheduling
Shifting Bottleneck ***Jm / / C_{max}***

- The solution is actually a little bit more complicated than before
- Precedence constraints exist because of other (already scheduled) machines
- *Delay precedence constraints* need to be added, if required

100

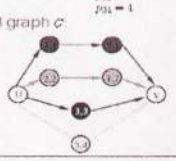
Shifting Bottleneck
Jm / / C_{max}

Example Delayed Precedences 1

Jobs 1: (1,1)→(2,1)
J2: (2,2)→(1,2)
J3: (3,1)
J4: (3,4)

Processing Times: $p_{11} = 1$, $p_{21} = 1$, $p_{22} = 1$, $p_{12} = 1$, $p_{32} = 4$, $p_{34} = 1$

Initial graph G:



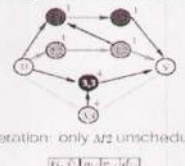
Example Delayed Precedences 2

After 2 iterations SBH we get:

$M_0 = \{M3, M1\}$

$(3,1) \rightarrow (3,3)$ and $(1,2) \rightarrow (1,1)$

Resulting graph G: $(C_{max}(M_0) = 9)$



3. iteration: only M2 unscheduled

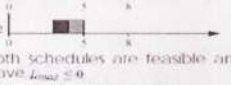
(i,j)	p_{ij}	r_{ij}	d_{ij}
(2,1)	1	3	8
(2,2)	1	0	5

101

Job Shop Scheduling
Shifting Bottleneck ***Jm / / C_{max}***

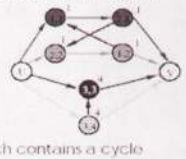
Example Delayed Precedences 3

Possible schedules for $M2$:



Both schedules are feasible and have $L_{max} \leq 0$

But: 2. schedule leads to



which contains a cycle

Delayed Precedences

The example shows:

- not all solutions of the one-machine problem fit to the given selections for machines from M_0
- the given selections for machines from M_0 may induce precedences for machines from $M \setminus M_0$

Example: scheduling operation (1,2) before (1,1) on machine M1 induces a delayed precedence constraint between (2,2) and (2,1) with length 3 → operation (2,1) has to start at least 3 time units after (2,2)

this time is needed to process operations (2,2), (1,2), and (1,1)

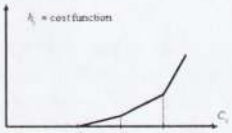
102

) m different machine
min $\sum(w_j T_j)$.

Job Shop Scheduling
Shifting Bottleneck ***Jm / / $\Sigma w_j T_j$***

Total Weighted Tardiness

- Need *n* sinks in disjunctive graph
- Machines scheduled one at a time
- Given current graph calculate completion time of each job
- Assess *n* due dates for each operation
- Piecewise linear penalty function



Job Shop Scheduling
Shifting Bottleneck ***Jm / / $\Sigma w_j T_j$***

Machine Selection

- Machine criticality
  - Solve a single machine problem
  - May have delayed precedence constraints
  - Generalizes single-machine with *n* jobs, precedence constraints, and total weighted tardiness objective
  - Apparent Tardiness Cost (ATC) Dispatching Rule

review.

Job Shop Scheduling
Shifting Bottleneck ***Jm / / $\Sigma w_j T_j$***

Apparent Tardiness Cost (ATC) Dispatching Rule

- Dynamic ranking index

$$I_j(t) = \frac{w_j}{p_j} \exp\left(- \frac{\max\{ d_j - p_j - t, 0 \}}{K \cdot \bar{p}(t)} \right)$$

Scaling constant

average p_j of the remaining jobs (rounded down)

- When machine becomes free:
  - Compute index for all remaining jobs that are currently waiting
  - Select job with highest value

Job Shop Scheduling
Shifting Bottleneck ***Jm / / $\Sigma w_j T_j$***

Choosing *K*

- Value of *K* determined empirically
- Related to the *due date tightness* factor $\tau = 1 - \frac{\bar{d}}{C_{max}}$
- and the *due date range* factor $R = \frac{d_{max} - d_{min}}{C_{max}}$
- Usually $1.5 \le K \le 4.5$

  $K = 4.5 + R$, *for* $R \le 0.5$

  $K = 6 - 2R$, *for* $R > 0.5$
- Rules of thumb:
  - Fix $K = 2$ for single machine or flow shop.
  - Fix $K = 3$ for dynamic job shops.

Job Shop Scheduling
Shifting Bottleneck ***Jm / / $\Sigma w_j T_j$***

Apparent Tardiness Cost with Setups (ATCS)

$$I_j(t) = \frac{w_j}{p_j} e^{\left(-\frac{\max(d_j - p_j - t, 0)}{k_1 \bar{p}(t)}\right)} e^{\left(-\frac{s_{ij}}{k_2 \bar{s}(t)}\right)}$$

k_1=due date scaling par.

k_2=setup time scaling par.

S: setups

k_1 and k_2 functions of:

- Due Date tightness $\tau = 1 - \bar{d}/C_{max}$
- Due Date Range $R = (d_{max} - d_{min})/C_{max}$
- Setup Time Severity $\eta = \bar{s}(t)/\bar{p}(t)$

$k_1 = 4.5 + R$, for $R \le 0.5$
$k_1 = 6 - 2R$, for $R > 0.5$
$k_2 = \tau/(2\sqrt{\eta})$

Job Shop Scheduling
Shifting Bottleneck ***Jm / / $\Sigma w_j T_j$***

Generalized ATC Rule

$$I_j(t) = \sum_{k=1}^{n} \frac{w_j}{p_j} \exp\left(- \frac{\left(d_{ij}^k - p_{ij} + (r_{ij} - t)\right)^+}{K \cdot \bar{p}(t)} \right)$$

Ranks jobs by impact on all *n* due dates

Job Shop Scheduling
Shifting Bottleneck *Jm | | $\Sigma w_j T_j$*

Criticality of Machines

- Criticality = subproblem m objective function

$$wT(m) = \sum_{j=1}^{n} w_j T_{m,j}$$

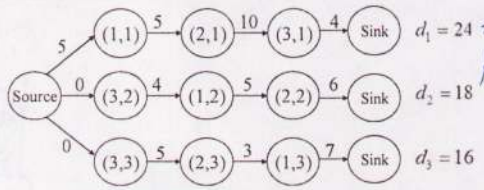
- More effective ways, e.g.
  - Add disjunctive arcs for each machine
  - Calculate new completion times C_k'' and

$$\sum_{k=1}^{n} w_k \left(C_k'' - C_k'\right) \cdot \exp\left(-\frac{\left(d_k - C_k'\right)^+}{K}\right)$$

109

Job Shop Scheduling
Shifting Bottleneck *Jm | | $\Sigma w_j T_j$*

Example



$w_1=1$, $w_2=2$, $w_3=2$

110

Job Shop Scheduling
Shifting Bottleneck *Jm | | $\Sigma w_j T_j$*

Subproblem: Machine 1

Jobs	1	2	3
p_{1j}	5	5	7
r_{1j}	5	4	8
$d_{1j}^1, d_{1j}^2, d_{1j}^3$	10,-,-	-,12,-	-,-,16

d_{ij} is the corresponding due date.

d_k 減 {(ij) 對 job k 的 Sink 的時間}

- is infinite. ∵ 不能對 job k.

111

Job Shop Scheduling
Shifting Bottleneck *Jm | | $\Sigma w_j T_j$*

Subproblem: Machine 2

Jobs	1	2	3
p_{2j}	10	6	3
r_{2j}	10	9	5
$d_{2j}^1, d_{2j}^2, d_{2j}^3$	20,-,-	-,18,-	-,-,9

112

Job Shop Scheduling
Shifting Bottleneck *Jm | | $\Sigma w_j T_j$*

Subproblem: Machine 3

Jobs	1	2	3
p_{3j}	4	4	5
r_{3j}	20	0	0
$d_{3j}^1, d_{3j}^2, d_{3j}^3$	24,-,-	-,7,-	-,-,6

113

Job Shop Scheduling
Shifting Bottleneck *Jm | | $\Sigma w_j T_j$*

- Solve using dispatching rule
  - Use K=0.1
  - Have t = 4,
- For machine 1 this results in

Jobs	1	2	3
p_{1j}	5	5	7
r_{1j}	5	4	8
$d_{1j}^1, d_{1j}^2, d_{1j}^3$	10,-,-	-,12,-	-,-,16

$\bar{p} = 5$

$I_{11}(4) = 1.2 \cdot 10^{-6}$

$I_{12}(4) = 3.3 \cdot 10^{-7}$

$I_{13}(4) = 1.5 \cdot 10^{-12}$

Schedule first

then calculate...

$I_{12}(10)$

$I_{13}(10)$

114

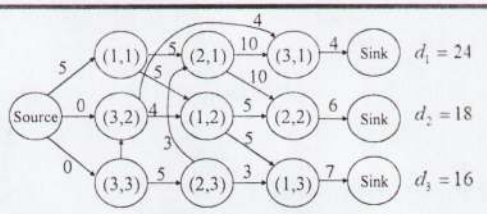
Job Shop Scheduling
Shifting Bottleneck ***Jm / / ΣwjTj***

Schedule first

Machine	Sequence	Criticality Value, wT(m)
1	(1,1),(1,2),(1,3)	18
2	(2,3),(2,1),(2,2)	16
3	(3,3),(3,2),(3,1)	4

Subproblem Solutions

Job Shop Scheduling
Shifting Bottleneck ***Jm / / ΣwjTj***



$C_{m,j}$ using forward pass of CPM

	m_1	m_2	m_3
j_1	10	20	24
j_2	15	26	9
j_3	22	8	5

118

all longest path.

Job Shop Scheduling
Shifting Bottleneck ***Jm / / ΣwjTj***

- Machine Criticality, *wT(m)*

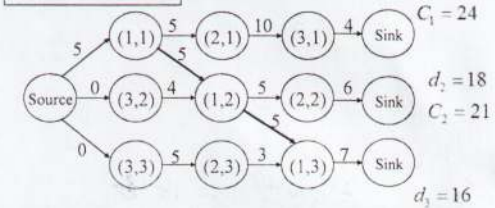
$$wT(1) = (1)(10-10) + (2)(15-12) + (2)(22-16) = 18$$
$$wT(2) = (1)(20-20) + (2)(26-18) + (2)(8-9) = 16$$
$$wT(3) = (1)(24-24) + (2)(9-7) + (2)(5-6) = 4$$

choose the biggest one.

117

Job Shop Scheduling
Shifting Bottleneck ***Jm / / ΣwjTj***

First Iteration



118

Job Shop Scheduling
Shifting Bottleneck ***Jm / / ΣwjTj***

Subproblem: Machine 2

Jobs	1	2	3
p_{2j}	10	6	3
r_{2j}	10	15	5
$d^1_{2j}, d^2_{2j}, d^3_{2j}$	20,-,-	-,21,-	-,-,15

119

$C_{ik} = \{(i,j) \text{ 到 } k \text{ 的 Sink 的 path}\}$

Job Shop Scheduling
Shifting Bottleneck ***Jm / / ΣwjTj***

Subproblem: Machine 3

Jobs	1	2	3
p_{3j}	4	4	5
r_{3j}	20	0	0
$d^1_{3j}, d^2_{3j}, d^3_{3j}$	24,-,-	-,10,10	-,-,12

120

Job Shop Scheduling
Shifting Bottleneck *Jm | | $\Sigma w_j T_j$*

Schedule first

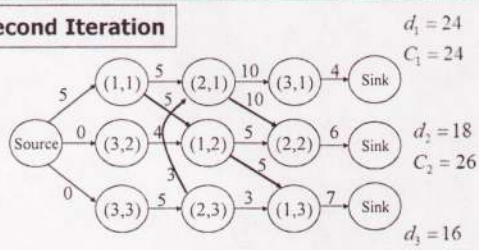
Machine	Sequence	wT(m) Value
2	(2,3),(2,1),(2,2)	10
3	(3,2),(3,3),(3,1)	0

Subproblem Solutions

121

Job Shop Scheduling
Shifting Bottleneck *Jm | | $\Sigma w_j T_j$*

Second Iteration



$d_1 = 24$
$C_1 = 24$
$d_2 = 18$
$C_2 = 26$
$d_3 = 16$
$C_3 = 22$

122

Job Shop Scheduling
Shifting Bottleneck *Jm | | $\Sigma w_j T_j$*

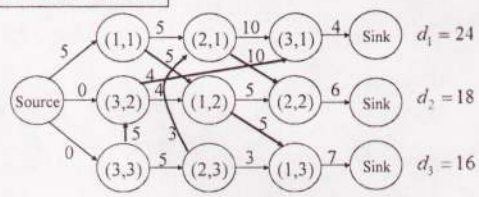
Subproblem: Machine 3

Jobs	1	2	3
p_{3j}	4	4	5
r_{3j}	20	0	0
$d^1_{3j}, d^2_{3j}, d^3_{3j}$	24,-,-	-,15,10	7,7,12

123

Job Shop Scheduling
Shifting Bottleneck *Jm | | $\Sigma w_j T_j$*

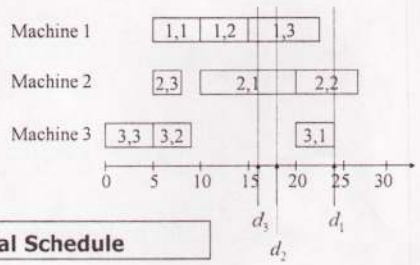
Third Iteration



$d_1 = 24$
$d_2 = 18$
$d_3 = 16$

124

Job Shop Scheduling
Shifting Bottleneck *Jm | | $\Sigma w_j T_j$*



Final Schedule

125

Job Shop Scheduling
Shifting Bottleneck *Jm | | $\Sigma w_j T_j$*

Performance

- Objective function

$$\sum_{j=1}^{3} w_j T_j = 1\cdot(24-24)^+ + 2\cdot(26-18)^+ + 2\cdot(22-16)^+ = 28$$

- A better schedule can be found using LEKIN (optimal value = 18)
- Reoptimization skipped in example... would have improved solution

126

Specialized Flow Shops: Flexible Assembly Systems

ISE 419

1

Flow Shop Scheduling

Limited Intermediate Storage *Flexible Assembly*

Job Shops	Flexible Assembly
❖ Each job has an unique identity	❖ Limited number of product types
❖ Make to order, low volume environment	❖ Given quantity of each type
❖ Possibly complicated route through system	❖ Mass production
	❖ High degree of automation
	❖ **Even more difficult!**
❖ Very difficult	

additional constraints that are imposed by the material handling systems.

2

Flow Shop Scheduling

Limited Intermediate Storage *Flexible Assembly*

- ❖ Flexible Manufacturing Systems (FMS)
  - Numerically Controlled machines
  - Automated Material Handling system
  - Produces a variety of product/part types
- ❖ Scheduling
  - Routing of jobs
  - Sequencing on machines
  - Setup of tools
- ❖ Similar features but more complicated

3

Flow Shop Scheduling

Limited Intermediate Storage *Flexible Assembly*

- ❖ Sequencing Unpaced Assembly Systems
  Simple flow line with finite buffers
  Application: assembly of copiers
  in series. max (throughput).
- ❖ Sequencing Paced Assembly Systems
  Conveyor belt moves at a fixed speed
  Application: automobile assembly
  more constraints. min (setup costs) and no overloaded.
- ❖ Scheduling Flexible Flow Systems
  Flow lines with finite buffers and bypass
  Application: producing printed circuit board
  machines in parallel at each workcenter.

max (throughput).

4

Flow Shop Scheduling

Sequencing Unpaced Assembly Systems

- ❖ Number of machines in series
- ❖ No buffers ← limited buffers.
- ❖ Material handling system
  - When a job finishes moves to next station
  - No bypassing
  - Blocking
- ❖ Can model any *finite buffer* situation — buffer space → a machine processing times is 0.
- ❖ Schedules often cyclic or periodic:
  - Given set of jobs scheduled in certain order
    - Contains all product types
    - May contain multiple jobs of same type
  - Second identical set scheduled, etc.

Cyclic schedules — repeated over and over again.

- ❖ Insignificant setup time
  - Low inventory cost
  - Easy to implement

5

Flow Shop Scheduling

Sequencing Unpaced Assembly Systems

Minimum Part Set MPS.

- ❖ Suppose l product types
- ❖ Let N_k be target number of jobs of type k
- ❖ Let z be the greatest common divisor
- ❖ Then $N^* = \left(\frac{N_1}{z}, \frac{N_2}{z}, \ldots, \frac{N_l}{z}\right)$

  is the smallest set with 'correct' proportions
- ❖ Example:

k	1	2	3	4
N_k	300	250	775	1000
N^*	300/25=12	250/25=10	775/25=31	1000/25=40

6

PS: Blocking is closing the machine for a while when there is no available job.

Flow Shop Scheduling
Sequencing Unpaced Assembly Systems

Defining a Cyclic Schedule

- Consider the jobs in the MPS as n jobs

$$n = \frac{1}{z}\sum_{k=1}^{l} N_k = 12+10+31+40 = 93$$

- A cyclic schedule is determined by sequencing the jobs in the MPS
- Maximizing Throughput = Minimizing cycle time

7

Flow Shop Scheduling
Sequencing Unpaced Assembly Systems

Sequencing Example

- Master Production Schedule calls for 30,000 units in the month (30 days)
  - 1000 per day
    - 3 shifts
      - 333 units per shift
    - 420 minutes per shift (60 minutes maintenance and capacity buffer)
      - 420/333 = 1.26 minutes per unit
- Breakdown by product proportion
  - Product X = 20000 units (66 percent)
  - Product Y = 5000 units (17 percent)
  - Product Z = 5000 units (17 percent)

8

Flow Shop Scheduling
Sequencing Unpaced Assembly Systems

- Proportional Schedule
  - X X Y X X Z X X Y X X Z X X Y X X Z.....
    - output rate - 1 unit each 1.26 minutes
    - X $\frac{(1.26)(2) + 2.52(1)}{3}$ = 1.68 minutes
    - Y (1.26)(6) = 7.56 minutes
    - Z (1.26)(6) = 7.56 minutes

9

Flow Shop Scheduling
Sequencing Unpaced Assembly Systems

Uniform Plant Loading

Weekly Production Required				
A	10 units			
B	20 units			
C	5 units			
D	5 units			
E	10 units			
Traditional Production Plan				
Monday	Tuesday	Wednesday	Thursday	Friday
AAAAA	BBBBB	BBBBB	DDDDD	EEEEE
AAAAA	BBBBB	BBBBB	CCCCC	EEEEE
JIT Plan with Level Scheduling				
Monday	Tuesday	Wednesday	Thursday	Friday
AABBBB	AABBBB	AABBBB	AABBBB	AABBBB
CDEE	CDEE	CDEE	CDEE	CDEE

10

Flow Shop Scheduling
Sequencing Unpaced Assembly Systems

Just-In-Time

JIT Requires Fast Setups

- SMED Concepts
  - internal setup
  - external setup
- Basic Steps to Setup Reduction
  - Identify all tasks as internal and external tasks
  - Convert as many tasks as possible to external
  - eliminate adjustments or reduce adjustment time
  - Eliminate the setup
    - standardize product requirements (engineering and manufacturing working together.)

11

Flow Shop Scheduling
Sequencing Unpaced Assembly Systems

Minimizing Cycle Time

- Profile Fitting (PF) heuristic:
  - Select first job j_1
    - Arbitrarily
    - Largest amount of processing
  - Generates profile

$$D_{i,j_1} = \sum_{k=1}^{i} p_{k,j_1}$$

departure times off the succession of machines

  - Determine which job goes next
- PF heuristic performs well in practice
- Basic assumptions
  - Setup is not important
  - Low WIP is important ⇒ Cyclic schedules good

12

Work-In-Process

find difference in (nwt) and (block)

no intermediate storage
two blocking 两个概念.
machine 中途停

Fm | block | Cmax

Flow Shop Scheduling

Limited Intermediate Storage ***Fm / block / C_{max}***

- If m_i finishes before m_{i+1}, job on m_i is *blocked*
- Flow shop with *finite* intermediate storage equivalent to a flow shop with *no* intermediate storage
  - each storage space can be considered a "machine" with processing time of zero for all jobs
  - *Fm / block / C_{max}* assumes no intermediate storage
- Job *departure time* calculations
  - D_{ij} = departure time of job j from m_i ≥ C_{ij}
  - D_{0j} = *start time* of job j on first machine

$$D_{i,j_1} = \sum_{i=1}^{i} p_{i,j_1}$$ — first job not blocked

current job off last machine

$$D_{i,j_k} = \max\left\{D_{i-1,j_k} + p_{i,j_k},\ D_{i+1,j_{k-1}}\right\}$$ — previous job on next machine

$$D_{m,j_2} = D_{m-1,j_1} + p_{m,j_1}$$ — no blocking on last machine

13

$D_{i,j_2} = \max\left(D_{i+1,j_2} + P_{i,j_2},\ D_{i+1,j_1}\right)$.

$D_{i,j_k} = \max\left\{D_{i+1,j_k} + P_{i,j_k},\ D_{i+1,j_{k-1}}\right\}$.

Flow Shop Scheduling

Sequencing Unpaced Assembly Systems

PF: Next Job

- Compute for each candidate job
  - Time machines are idle
  - Time job is blocked
- Start with departure times:

$$D_{1,j_2} = \max\left(D_{1,j_1} + p_{1c},\ D_{2,j_1}\right)$$

$$D_{i,j_2} = \max\left(D_{i-1,j_2} + p_{ic},\ D_{i+1,j_1}\right), \quad i = 2,\ldots,m-1$$

$$D_{m,j_2} = \max\left(D_{m-1,j_2} + p_{mc}\right)$$

14

Flow Shop Scheduling

Sequencing Unpaced Assembly Systems

Nonproductive Time

PF

- Calculate sum of idle and blocked time

$$I_c = \sum_{i=1}^{m} I_{i,c} = \sum_{i=1}^{m}\left(D_{i,j_2} - D_{i,j_1} - p_{i,c}\right)$$

  - Repeat for all remaining jobs
  - Select job with smallest number
  - Calculate new profile and repeat
- *Incorporating "bottleneckness" of machine*
  - let $w_i = \sum_j p_{i,j}$
  - compare weighted sum of wasted time $\sum_{i=1}^{m} w_i I_{i,c}$ to select next job k

15

每个 machine 的 idle 时间.

WPF.

weighted profile fitting.

Flow Shop Scheduling

Sequencing Unpaced Assembly Systems

- Example

j	1	2	3	4
$p_{1,j}$	2	4	2	3
$p_{2,j}$	4	4	0	2
$p_{3,j}$	2	0	2	0
$\Sigma p_{i,j}$	8	8	4	5

jobs 1 and 2 offer most load ... pick job 1

- Job 1 "profile":

$D_{i,j}$	[1]=1
i=1	2
i=2	6
i=3	8

the first one is job 1.

16

Flow Shop Scheduling

Sequencing Unpaced Assembly Systems

- Assess candidate jobs for position [2]:

$D_{i,j}$	[1]=1	c=2	c=3	c=4
i=1	2	{2+4=6, 6}=**6**	{2+2=4, 6}=**6**	{2+3=5, 6}=**6**
i=2	6	{6+4=10, 8}=**10**	{6+0=6, 8}=**8**	{6+2=8, 8}=**8**
i=3	8	10+0=**10**	8+2=**10**	8+0=**8**
I_c		26-16-8=**2**	24-16-4=**4**	22-16-5=**1**

$D_{i,j_2} = \max\left(D_{i-1,j_2} + p_{ic},\ D_{i+1,j_1}\right)$ — let [2]=4

17

$\Sigma D_{c2} - \Sigma D_1 - \Sigma P_{c-2}$.

the second one is job 4

Flow Shop Scheduling

Sequencing Unpaced Assembly Systems

- Assess candidate jobs for position [3]:

$D_{i,j}$	[2]=4	c=2	c=3
i=1	6	{6+4=10, 8}=**10**	{6+2=8, 8}=**8**
i=2	8	{10+4=14, 8}=**14**	{8+0=8, 8}=**8**
i=3	8	14+0=**14**	8+2=**10**
I_c		38-22-8=**8**	26-22-4=**0**

S = 1-4-3-2

$D_{i,j_3} = \max\left(D_{i-1,j_3} + p_{ic},\ D_{i+1,j_2}\right)$ — let [3]=3

18

Flow Shop Scheduling

Sequencing Unpaced Assembly Systems

❖ Assess delay for position [4]=2 :

$D_{i,3}$	[3]=3	[4]=2
i=1	8	{8+4=12, 8}=**12**
i=2	8	{12+4=16, 10}=**16**
i=3	10	16+0=**16**
I_c		44-26-8=**10**

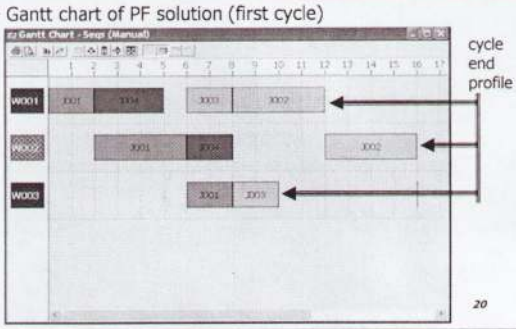
19

1-4-3-2

Flow Shop Scheduling

Sequencing Unpaced Assembly Systems

❖ Gantt chart of PF solution (first cycle)

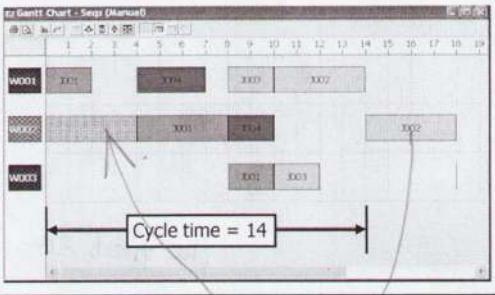


20

Flow Shop Scheduling

Sequencing Unpaced Assembly Systems

❖ S = 1-4-3-2 with machine 2 ready time = 4 :



21

Flow Shop Scheduling

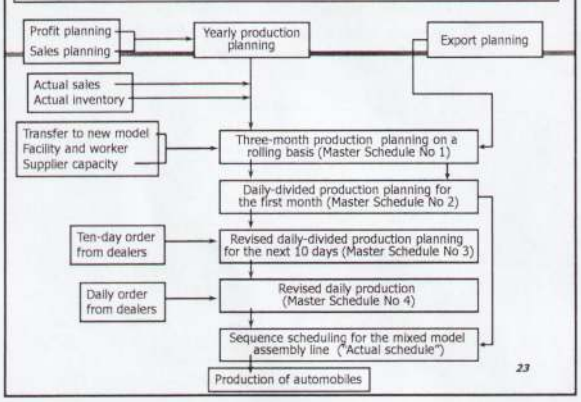
Limited Intermediate Storage ***Flexible Assembly***

Paced Assembly Systems

- ❖ Conveyor moves jobs at fixed speeds
- ❖ Fixed distance between jobs
  - Spacing proportional to processing time
- ❖ No bypass
- ❖ Unit cycle time
  - time between two successive jobs

22

Production Planning System in Nissan



23

Flow Shop Scheduling

Paced Assembly Systems

Grouping and Spacing

- ❖ Attributes and characteristics of each job
  - color, options, destination of cars
- ❖ Changeover cost
  - Group operations with high changeover
- ❖ Certain long operations
  - Space evenly over the sequence
  - *Capacity constrained operations* (criticality index)

24

Flow Shop Scheduling
Paced Assembly Systems

Objectives

- Minimize total setup cost
- Meet due dates for make-to-order jobs
  - Total weighted tardiness
- Spacing of capacity constrained operations
  - $P_i(l)$ = penalty for working on two jobs l positions apart in ith workstation (penalty decreasing in l)
- Regular (smooth) rate of material consumption

$\min(\sum P_i(l))$

part of it.

25

Flow Shop Scheduling
Paced Assembly Systems

Grouping and Spacing Heuristic GS

1. Determine the total number of jobs to be scheduled
   - ... higher number, more opportunity to lower scheduling costs
   - ... lower number lowers disruptive effect of downtime
2. Group jobs with high setup cost operations
3. Order each subgroup accounting for shipping dates
4. Space jobs within subgroups accounting for capacity constrained operations

4 steps.

26

obj: $\min\left(\sum c_{jk} + \sum \text{spacing costs} + \sum \text{tardiness costs}\right)$

$\sum P(l)$. $\sum w_i T_j$

Flow Shop Scheduling
Paced Assembly Systems

Example

- Single machine with 10 jobs
- Each job has a unit processing time
- Two parameters
  - a_{k1} relates to setup groupings
  - a_{k2} relates to "capacity constrained" operations

attributes

Setup cost: $c_{jk} = |a_{j1} - a_{k1}|$

Criticality index.

in the step 2.

If $a_{j2} = a_{k2}$ there is a penalty cost

$P_2(l) = \max(3 - l, 0)$

in the step 4.

l: positions apart in ith workstation.

27

Flow Shop Scheduling
Paced Assembly Systems

Example Data

Job	1	2	3	4	5	6	7	8	9	10
a_{j1}	1	1	1	3	3	3	5	5	5	5
a_{j2}	0	1	1	0	1	1	1	0	0	0
d_j	∞	2	∞	∞	∞	∞	6	∞	∞	∞
w_j	0	4	0	0	0	0	4	0	0	0

maybe color.

28

a 越大, operation will be longer,
大招之间冷却时间长, 强行连续施放就要买刷新道具. penalty cost.

step2. **Flow Shop Scheduling**
Paced Assembly Systems

Grouping

- Group A: Jobs 1,2, and 3
- Group B: Jobs 4,5, and 6
- Group C: Jobs 7,8,9, and 10

- Best order: A → B → C

$c_{AB} = c_{BC} = |a_{j1} - a_{k1}| = 2$

with regard to $\min(\sum C_{jk})$.

29

step3. **Flow Shop Scheduling**
Paced Assembly Systems

	A			B			C			
Job	1	2	3	4	5	6	7	8	9	10
a_{j1}	1	1	1	3	3	3	5	5	5	5
a_{j2}	0	1	1	0	1	1	1	0	0	0
d_j	∞	2	∞	∞	∞	∞	6	∞	∞	∞
w_j	0	4	0	0	0	0	4	0	0	0

Due date

⇒ Order A → C → B

Grouped Jobs

may be tardiness penalty

$\sum C_{jk} = |5-1| + |3-5| = 6$.

step 4. **Flow Shop Scheduling**
Paced Assembly Systems

	A			C				B		
Job	2	1	3	8	7	9	10	5	4	6
a_{j1}	1	1	1	5	5	5	5	3	3	3
a_{j2}	1	0	1	0	1	0	0	1	0	1
d_j	2	∞	∞	∞	6	∞	∞	∞	∞	∞
w_j	4	0	0	0	4	0	0	0	0	0

Capacity Constrained Operations

31

to $\min\left[\sum P_2(t)\right]$.

$= (3-2) + (3-2) + (3-3) + (3-2)$

$= 3.$

Flow Shop Scheduling
Paced Assembly Systems

MANAGEMENT SCIENCE © 2001 INFORMS
Vol. 47, No. 3, March 2001 pp. 486–491

Sequencing JIT Mixed-Model Assembly Lines Under Station-Load and Part-Usage Constraints

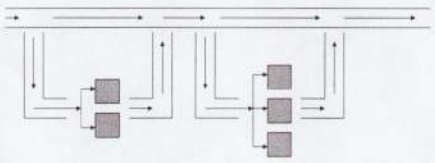
Andreas Drexl • Alf Kimms
Institut für Betriebswirtschaftslehre, Christian-Albrechts-Universität zu Kiel, Olshausenstr. 40, 24118 Kiel, Germany

This paper deals with two most important problems, from both practical and theoretical standpoints, arising in sequencing mixed-model assembly lines. Such lines have become core components of modern repetitive manufacturing, and notably in just-in-time (JIT) manufacturing in particular. One problem is to keep the usage rate of all parts fed into the final assembly as constant as possible (the "level-scheduling problem"), while the other is to keep the line's workstation loads as constant as possible (the "car-sequencing problem"). In this paper the combined problem is formulated as a single-integer programming model. The LP relaxation of this model is solved by column-generation techniques. The results of an experimental evaluation show that the lower bounds are tight.

32

Flow Shop Scheduling
Limited Intermediate Storage *Flexible Assembly*

Flexible Flow System with Bypass



❖There may be multiple machines at each station and/or there may be bypass

33

Flow Shop Scheduling
Flexible Flow System with Bypass

Flexible Flow Line Algorithm FFLL

- ❖ Objectives
  - Maximize throughput
  - Minimize work-in-process (WIP)
- ❖ Minimizes the makespan of a day's mix
  - Actually minimization of cycle time for MPS
- ❖ Reduces blocking probabilities
- ❖ Three phases:
  - Machine allocation phase
    - assigns each job to a specific machine at station
  - Sequencing phase
    - order in which jobs are released
    - dynamic balancing heuristic
  - Time release phase
    - minimize MPS cycle time on bottlenecks

34

Flow Shop Scheduling
Flexible Flow System with Bypass

(1) **Machine Allocation**

- ❖ Bank of machines
  - Which machine for which job?
- ❖ Basic idea: workload balancing — LPT.
  - Use LPT dispatching rule

(2) **Sequencing**

- ❖ Dynamic balancing heuristic
  - Basic idea: spread out jobs sent to the same machine
- ❖ For a given station, let p_{ij} be processing time of job j on ith machine
- ❖ Let $W_i = \sum_{j=1}^{n} p_{ij}$ $\quad W = \sum_{i=1}^{m} W_i$

35

Workload $W_i = \sum_{j=1}^{n} P_{ij}$ $\quad W = \sum_{i=1}^{m} W_i$.

Flow Shop Scheduling
Flexible Flow System with Bypass

Dynamic Balancing Heuristic

- ❖ Let S_j be the jobs released before and including job j
- ❖ Define the fraction of total workload at machine i by the time job j enters

$$\alpha_{ij} = \sum_{k \in S_j} \frac{p_{ik}}{W_i} \in [0,1]$$

- ❖ Target for all jobs in S_j

$$\alpha_j^* = \sum_{k \in S_j} \sum_{i=1}^{m} p_{ik} \Big/ \sum_{k=1}^{n} \sum_{i=1}^{m} p_{ik} = \sum_{k \in S_j} p_k / W$$

36

Flow Shop Scheduling

Flexible Flow System with Bypass

Minimizing Overload

- Define the *overload* of the ith machine

$$o_{ij} = p_{ij} - p_j \frac{W_i}{W}$$

$$p_j = \sum_{i=1}^{m} p_{ij}$$

- The *cumulative overload* is

$$O_{ij} = \sum_{k \in S_j} o_{ik} = \sum_{k \in S_j} p_{ik} - \alpha_j W_i$$

Target workload on machine i

Actual workload on machine i

- Minimize

$$\sum_{i=1}^{m} \sum_{j=1}^{n} \max\{O_{ij}, 0\}$$

Sum overloads

37

find $\sum_{i=1}^{m} \max\{O_{ij}, 0\}$ to be the next job.

Flow Shop Scheduling

Flexible Flow System with Bypass

3 Release Timing

- MPS workload of each machine known
  - Highest workload = bottleneck
  - MPS cycle time = Bottleneck cycle time
- Algorithm
  - Step 1: Release all jobs as soon as possible
  - Step 2: Delay all jobs upstream from bottleneck as much as possible
  - Step 3: Move along all jobs downstream from the bottleneck as much as possible

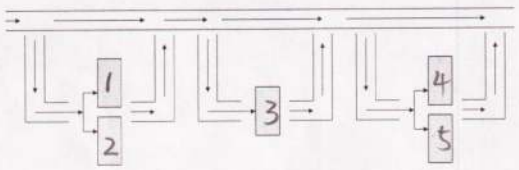
without altering the job sequences !

up | bottleneck | down

Flow Shop Scheduling

Flexible Flow System with Bypass

Example



39

Flow Shop Scheduling

Flexible Flow System with Bypass

Data

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

40

Flow Shop Scheduling

Flexible Flow System with Bypass

step 1

Machine Allocation: LPT assignment to machines at a stage

as several machines in parallel

Jobs	1	2	3	4	5
p_{1j}	6	0	0	3	0
p_{2j}	0	3	1	0	5
p_{3j}	3	2	1	3	2
p_{4j}	4	5	0	3	0
p_{5j}	0	0	6	0	4

41

Flow Shop Scheduling

Flexible Flow System with Bypass

step 2

Workload

$$W_i = \sum_{j=1}^{n} p_{ij} \qquad p_j = \sum_{i=1}^{m} p_{ij}$$

$W_1 = 9$ $p_1 = 13$

$W_2 = 9$ $p_2 = 10$

$W_3 = 11$ $p_3 = 8$

$W_4 = 12$ $p_4 = 9$

$W_5 = 10$ $p_5 = 11$

$W = 51$

42

identical

Flow Shop Scheduling
Flexible Flow System with Bypass

Calculate Target Workload

Jobs	1	2	3	4	5	W_i
p_{1j}	6	0	0	3	0	9
p_{2j}	0	3	1	0	5	9
p_{3j}	3	2	1	3	2	11
p_{4j}	4	5	0	3	0	12
p_{5j}	0	0	6	0	4	10
α_k^*	0.255	0.196	0.157	0.176	0.216	
	13/51	10/51	8/51	9/51	11/51	

W = 51

$$\alpha_j^* = \sum_{k\in J_j}\sum_{i=1}^{n_k} p_{ik} \Big/ \sum_{k=1}^{n}\sum_{i=1}^{n_k} p_{ik} = \sum_{k\in J_j} p_k / W$$

43

Flow Shop Scheduling
Flexible Flow System with Bypass

Overload Matrix

Jobs	1	2	3	4	5	W_i
p_{1j}	6	0	0	3	0	9
p_{2j}	0	3	1	0	5	9
p_{3j}	3	2	1	3	2	11
p_{4j}	4	5	0	3	0	12
p_{5j}	0	0	6	0	4	10
α_k^*	0.255	0.196	0.157	0.176	0.216	

W = 51

$O_{11} = 6 - 0.255 * 9 = 3.71$

$$O_{ik} = \left(\sum_{j\in J_k} p_{ij}\right) - \alpha_k^* W_i$$

44

without thinking about release time.

Assume: All jobs release at the same time. beggining.

Flow Shop Scheduling
Flexible Flow System with Bypass

Overload Matrix

Jobs	1	2	3	4	5	W_i
p_{1j}	6	0	0	3	0	9
p_{2j}	0	3	1	0	5	9
p_{3j}	3	2	1	3	2	11
p_{4j}	4	5	0	3	0	12
p_{5j}	0	0	6	0	4	10
α_k^*	0.255	0.196	0.157	0.176	0.216	

W = 51

$O_{11} = 6 - 0.25 * 9 = 3.75$

$O_{12} = 2 - 0.196 * 11 = -0.16$

$$O_{ik} = \left(\sum_{j\in J_k} p_{ij}\right) - \alpha_k^* W_i$$

45

Flow Shop Scheduling
Flexible Flow System with Bypass

Overload: job contribution to each as the first job in sequence

$$o_{11} = 6 - 13\cdot\frac{9}{51} = +3.71$$

$$o_{21} = 0 - 13\cdot\frac{9}{51} = -2.29$$

$$o_{31} = 3 - 13\cdot\frac{11}{51} = +0.20$$

$$o_{41} = 4 - 13\cdot\frac{12}{51} = +0.94$$

$$o_{51} = 0 - 13\cdot\frac{10}{51} = -2.55$$

46

Flow Shop Scheduling
Flexible Flow System with Bypass

Overload Matrix

$$O_{ik} = \left(\sum_{j\in J_k} p_{ij}\right) - \alpha_k^* W_i$$

Jobs	1	2	3	4	5	W_i
O_{1j}	3.71	-1.76	-1.41	1.41	-1.94	9
O_{2j}	-2.29	1.24	-0.41	-1.59	3.06	9
O_{3j}	0.20	-0.16	-0.73	1.06	-0.37	11
O_{4j}	0.94	2.65	-1.88	0.88	-2.59	12
O_{5j}	-2.55	-1.96	4.43	-1.76	1.84	10
α_k^*	0.255	0.196	0.157	0.176	0.216	

W = 51

47

Flow Shop Scheduling
Flexible Flow System with Bypass

Dynamic Balancing

3.71	0.00	0.00	1.41	0.00
0.00	1.24	0.00	0.00	3.06
0.20	0.00	0.00	1.06	0.00
0.94	2.65	0.00	0.88	0.00
0.00	0.00	4.43	0.00	1.84
4.84	3.88	4.43	3.35	4.90

First Job

48

Flow Shop Scheduling
Flexible Flow System with Bypass

Calculate Target Workload

Jobs	1	2	3	4	5	W_i
p_{1j}	6	0	0	3	0	9
p_{2j}	0	3	1	0	5	9
p_{3j}	3	2	1	3	2	11
p_{4j}	4	5	0	3	0	12
p_{5j}	0	0	6	0	4	10
α_k^*						

W = 51

Since job 4 goes, the meaning of each cell changes: total load up to (and including) job j

Consider the cumulative load on m1 if we choose job 1 next

assume : job 4 releases firstly.
and then job 1, 2, 3, 5

Flow Shop Scheduling
Flexible Flow System with Bypass

Selecting the Second Job

- Calculate the cumulative overload

$$O_{11} = \sum_{k \in S_j} p_{1k} - \alpha_1^* W_1$$
$$= \sum_{k \in \{4,1\}} p_{1k} - 0.43 \cdot 9$$
$$= (3+6) - 0.43 \cdot 9$$
$$= 5.13$$

where

$$\alpha_1^* = \sum_{k \in S_j} p_k / W = \sum_{k \in \{4,1\}} p_k / 51$$
$$= (9+13)/51 = 0.43$$

be careful !!!

like $\left(\sum_{i=1}^{n} P_{ij}\right)$.

Add the things of sequenced scheduled jobs

50

Flow Shop Scheduling
Flexible Flow System with Bypass

Calculate Target Workload

Jobs	1	2	3	4	5	W_i
p_{1j}	9	3	3	3	3	9
p_{2j}	0	3	1	0	5	9
p_{3j}	6	5	4	3	5	11
p_{4j}	7	8	3	3	3	12
p_{5j}	0	0	6	0	4	10
α_k^*	0.43	0.37	0.33		0.39	
	22/51	19/51	17/51		20/51	

W = 51

51

Flow Shop Scheduling
Flexible Flow System with Bypass

Jobs	1	2	3	4	5	W_i
O_{1j}	5.13	-0.33	0	3	-0.50	9
O_{2j}	-3.87	-0.33	2	0	1.42	9
O_{3j}	1.27	0.93	0.37	3	0.71	11
O_{4j}	1.84	3.56	-1	3	-1.50	12
O_{5j}	-4.3	-3.7	2.67	0	0.1	10
α_k^*	0.43	0.37	0.33		0.39	
$\sum_{i=1}^{n} \max(O_{ik}, 0)$	8.67	4.86	5.37		2.63	

W = 51

Cumulative Overload

Job 5 goes next

52

step 3:

Flow Shop Scheduling
Flexible Flow System with Bypass

Final Cycle

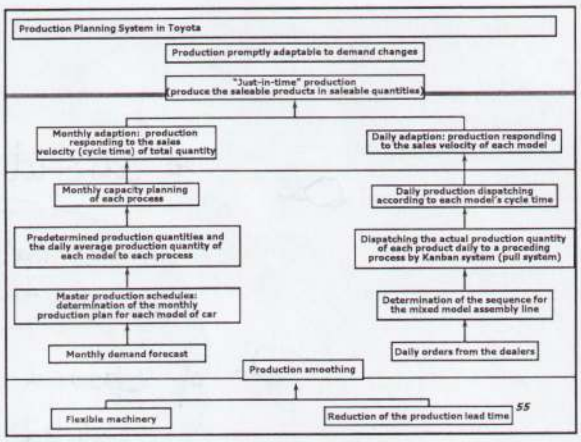
- Schedule jobs 4,5,1,3,2
- Release timing phase
  - Machine 4 is the bottleneck
  - Delay jobs on Machine 1, 2, and 3
  - Expedite jobs on Machine 5

53

Flow Shop Scheduling
Mixed Model Assembly Sequencing at Toyota

- Everything operates on Just-in-Time
  - Works to minimize WIP & tardiness
- To do this, the most important objective is to keep the part consumption regular
  - The quantity of a given part consumed per hour should be constant
- Solution: the next car to build is chosen to minimize error from the desired rate

54



Flow Shop Scheduling

Mixed Model Assembly Sequencing at Toyota

Example

- Model 1 requires 1 grommit
- Model 2 requires 2 grommits
- Model 3 requires 3 grommits
- We will produce:
  - 10 cars of model 1
  - 15 cars of model 2
  - 10 cars of model 3

56

Flow Shop Scheduling

Mixed Model Assembly Sequencing at Toyota

- So we need
  - (10 * 1) + (15 * 2) + (10 * 3) = 70
- If the consumption of grommits is to be constant, we need to consume 70/35 = 2 per car
- Choose car [k] to minimize:
  - (Grommits use/cars built – 2)^2
- Toyota tracks about 20 "parts" (subassemblies)
- Minimizes the sum of squared error over all these parts

57

Flow Shop Scheduling

Mixed Model Assembly Sequencing at Toyota

- "Rate-based" schedule
- Keep rate of part consumption as *regular* as possible
- Definitions:
  - N = total number of assemblies required
  - t = total number of assembly types
  - N_j = total number of assembly j required
  - m = total number of part types required for N assemblies
  - $b_{i,j}$ = number of part i needed for each assembly j
  - R_i = total number of part i required for N assemblies
  - $x_{i,k}$ = total number of part i required for first k assemblies in sequence

$$N = \sum_{j=1}^{t} N_j$$

58

Flow Shop Scheduling

Mixed Model Assembly Sequencing at Toyota

- Smoothing part consumption is achieved through position k in assembly sequence by minimizing:

$$\sum_{i=1}^{m}\left(\frac{kR_i}{N} - x_{i,k}\right)^2$$

goal — actual

- If assembly j is being considered for position k , define

$$x_{i,k} = x_{i,k-1} + b_{i,j}$$

59

Flow Shop Scheduling

Mixed Model Assembly Sequencing at Toyota

- *Goal-Chasing Algorithm*

(1) set $x_{i,0} = 0$, $S_0 = \{1, 2, \ldots, t\}$, $k = 1$

(2) put assembly j^* in position k if j^* satisfies

$$\min_{j \in S_{k-1}} \left\{ \sum_{i=1}^{m} \left(\frac{kR_i}{N} - x_{i,k-1} - b_{i,j} \right)^2 \right\}$$

(3) set $N_{j^*} = N_{j^*} - 1$

if $N_{j^*} > 0$, set $S_k = S_{k-1}$

else, $S_k = S_{k-1} - \{j^*\}$

(4) if $S_k = \varnothing$ ... Stop

else, set $x_{i,k} = x_{i,k-1} + b_{i,j^*}$ for $i = 1, \ldots, m$

set $k = k + 1$ and go to (2)

60

Economic Lot Scheduling Problem

IE 419

1

Economic Lot Scheduling
Single Item, Single Machine

Continuous manufacturing, make-to-stock:

- Domain:
  - large number of identical jobs
  - setup time/cost significant
  - setup may be sequence dependent
  - Longer planning horizon
  - Long runs, large setup times: cost minimization
  - Usually repetitive structure: rotation schedule
- Degrees of freedom: cycle length, sequencing within cycle
- Intuition: increasing lot size → run length
  - inventory costs per unit of time ↑
  - setup costs per unit of time ↓

2

refer jobs to items.
the interrupted processing of a series of identical items : a run.
produce identical items in long runs, tend to be Make-To-Stock.

Economic Lot Scheduling
Single Item, Single Machine

Standard ELSP:

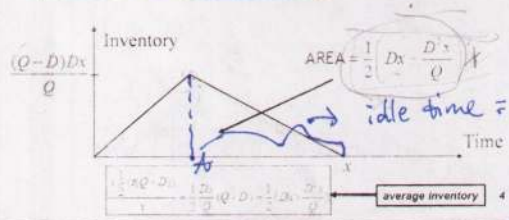
- 1 machine: produces item j with rate Q_j per time unit.
- n items with processing time $p_j \rightarrow Q_j = 1/p_j$
- constant demand rate D_j
- inventory costs h_j, setup cost c_{jk} (j → k), setup time s_{jk}
- cycle: $j_1, \ldots, j_n$, corresponding run times: $t_{j1}, \ldots, t_{jn}$

3

Economic Lot Scheduling
Single Item, Single Machine

Minimize Cost

- Let x denote the cycle time
- Demand over a cycle = Dx
- Length of production run needed = $Dx/Q = t$



has been determined.
idle time = $x - t$
$= x\left(1 - \frac{D}{Q}\right)$.
$\rho = \frac{D}{Q}$ is utilization.

4

Economic Lot Scheduling
Single Item, Single Machine

- Setup cost rate = c/x ← Setup cost per run
- Average inventory holding cost

$$\frac{1}{2}h\left(Dx - \frac{D^2x}{Q}\right)$$

Per item holding cost

- Total cost

$$\frac{1}{2}h\left(Dx - \frac{D^2x}{Q}\right) + \frac{c}{x}$$

the total average cost per time unit

5

Economic Lot Scheduling
Single Item, Single Machine

Optimizing Cost

- Derivative

$$\frac{d}{dx}\left[\frac{1}{2}h\left(Dx - \frac{D^2x}{Q}\right) + \frac{c}{x}\right] = \frac{1}{2}hD\left(1 - \frac{D}{Q}\right) - \frac{c}{x^2}$$

- Solve

$$\frac{1}{2}hD\left(1 - \frac{D}{Q}\right) - \frac{c}{x^2} = 0 \rightarrow \frac{1}{2}hD\left(1 - \frac{D}{Q}\right) = \frac{c}{x^2}$$

$$x = \sqrt{\frac{2Qc}{hD(Q-D)}}$$

6

Economic Lot Scheduling
Single Item, Single Machine

Optimal Lot Size

- Total production

$$Ds = \sqrt{\frac{2QcD}{h(Q-D)}}$$

- When unlimited production capabilities

$$\sqrt{\frac{2QcD}{h(Q-D)}} \xrightarrow{Q\to\infty} \sqrt{\frac{2cD}{h}}$$

– *Economic Order Quantity (EOQ)*

Economic Lot Scheduling
Single Item, Single Machine

Setup Time

- Setup time s
- If $s \le x(1-\rho)$ ...lot size is optimal
- Otherwise cycle length is optimal

$$\rho = \frac{q}{Q} = \text{utilization}$$

$$x = \frac{s}{1-\rho}$$

Economic Lot Scheduling
Single Item, Single Machine

Example

- Production Q = 90/month
- Demand D = 50/month
- Setup cost c = \$2000
- Holding cost h = \$20/item

$$x = \sqrt{\frac{2 \cdot 90 \cdot 2000}{20 \cdot 50(90-50)}} = \sqrt{\frac{3600}{10 \cdot 40}} = \sqrt{\frac{36}{4}} = 3$$

- Cycle time = 3 months
- Lot size = 150 items
- Idle time = 3(1-5/9)=1.33 months

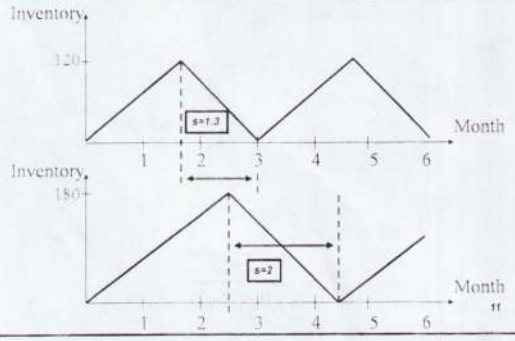
Economic Lot Scheduling
Single Item, Single Machine

Example: Setup Times

- Now assume setup time
- If setup time < 1.33 months
  – then 3 month cycle *still optimal*
- *Otherwise* the cycle time must be longer

$$x = \frac{s}{1-\frac{D}{Q}} = \frac{s}{1-\rho}$$

Economic Lot Scheduling
Single Item, Single Machine



Economic Lot Scheduling
Multiple Items, Single Machine

Multiple Items

- Now assume n different items
- Demand rate for item j is D_j
- Production rate of item j is Q_j
- Setup independent of the sequence
- **Rotation schedule**: single run of each item
  – Cycle length determines the run length for each item
  – Only need to determine the cycle length x
  – Expression for total cost/time unit

Economic Lot Scheduling
Multiple Items, Single Machine

- Average inventory level for the j-th item

$$\frac{1}{2}\left(D_j x - \frac{D_j^2 x}{Q_j}\right)$$

- Average total cost per unit time.

$$\sum_{j=1}^{n}\left(\frac{1}{2}h_j\left(D_j x - \frac{D_j^2 x}{Q_j}\right) + \frac{c_j}{x}\right)$$

No setup time

13

Economic Lot Scheduling
Multiple Items, Single Machine

Optimal Lot-Size

- Solve as before (derivative.)

$$x = \sqrt{\left(\sum_{j=1}^{n}\frac{h_j D_j (Q_j - D_j)}{2Q_j}\right)^{-1}\sum_{j=1}^{n} c_j}$$

- Limiting case (infinite production rate) $Q_j \to \infty$.

$$x = \sqrt{\left(\sum_{j=1}^{n}\frac{h_j D_j}{2}\right)^{-1}\sum_{j=1}^{n} c_j}$$

14

Economic Lot Scheduling
Multiple Items, Single Machine

Feasibility check

- Feasibility
  - $\Leftrightarrow$ Q > D (demand can be met)
  - $\Leftrightarrow$ D/Q < 1
  - $\Leftrightarrow$ $\rho < 1$ (ρ=machine utilization).

When n>1:

$$\rho = \sum_j \rho_j < 1.$$

15

Economic Lot Scheduling
Multiple Items, Single Machine

Example

Items	1	2	3	4
Q_j	400	400	500	400
D_j	50	50	60	60
h_j	20	20	30	70
c_j	2000	2500	800	0

16

Economic Lot Scheduling
Multiple Items, Single Machine

Solution

$$x = \sqrt{\left(\sum_{j=1}^{n}\frac{h_j D_j (Q_j - D_j)}{2Q_j}\right)^{-1}\sum_{j=1}^{n} c_j}$$

$$= \sqrt{\left(2\cdot\frac{10\cdot 350}{8} + \frac{18\cdot 440}{10} + \frac{42\cdot 340}{8}\right)^{-1} 5300}$$

$$= \sqrt{\left(\frac{10\cdot 350}{4} + \frac{18\cdot 440}{10} + \frac{42\cdot 340}{8}\right)^{-1} 5300}$$

$$= \sqrt{(3452)^{-1}\,5300} = \sqrt{1.5353} = 1.24 \text{ months}$$

17

Economic Lot Scheduling
Multiple Items, Single Machine

$$\sum_{j=1}^{n}\left(\frac{1}{2}h_j\left(D_j x - \frac{D_j^2 x}{Q_j}\right) + \frac{c_j}{x}\right)$$

- The total average cost per time unit is

$$2155 + 2559 + 1627 + 2213 = \$8554$$

- How can we do better than this?

18

① No setup time ↑.

Economic Lot Scheduling
Multiple Items, Single Machine

Setup Times

- With sequence independent setup costs and no setup times the sequence within each lot does not matter

  ⇒ *Only a lot sizing problem*

- Even with setup times, if they are not job dependent then still only lot sizing

19

Economic Lot Scheduling
Multiple Items, Single Machine

Job Independent Setup Times

- If sum of setup times < idle time then our optimal cycle length remains optimal
- Otherwise we take it as small as possible

$$x = \frac{\sum_{j=1}^{n} s_j}{1 - \sum_{j=1}^{n} \rho_j} \quad \rightarrow \text{setup time}$$

$\approx D_j \cdot Q_j$

20

Idle time

Cycle time

Time (months)

Economic Lot Scheduling
Multiple Items, Single Machine

Job Dependent Setup Times

- *Solution OK if:* for any k, $\sum s_{jk} < (1 - \sum \rho_j)x^*$ = idle time
- Otherwise, there is a *sequencing problem*
- *Objective:* minimize sum of setup times
- in case of sequence dependent setup times, s_{jk} minimized by solving corresponding TSP, denote minimum by T

  Note: there are (n-1)! different sequences (rotation!!)
- if $T = \min \sum s_{jk} > (1 - \sum \rho_j)x^*$ then increase x until:

$$T = \min \sum s_{jk} = (1 - \sum \rho_j)x$$

$$\Rightarrow x^* = \frac{\sum s_{jk}}{(1 - \sum \rho_j)}$$ (i.e., machine is never idle)

21

An Integer Programming Formulation of the TSP

$$\text{minimize} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} s_{ij} x_{ij}$$

subject to

$$\sum_{j=0}^{n} x_{ij} = 1 \qquad 1 \le i \le n$$

$$\sum_{i=0}^{n} x_{ij} = 1 \qquad 1 \le j \le n$$

$$t_i - t_j + n x_{ij} \le n-1 \qquad 2 \le i, j \le n$$

- s_{ij} = setup time between job i and job j
- x_{ij} = 1 if "tour" visits i then j, and 0 otherwise (binary)
- t_i = *arbitrary* real numbers we need to solve for

22

Heuristics for the TSP

To apply the nearest-neighbor heuristic (NNH)

- We begin at any city and then "visit" the nearest city.
- Then we go to the unvisited city closest to the city we have most recently visited.
- We continue in this fashion until a tour is obtained.
- After applying this procedure beginning at each city, we take the best tour found.

In the cheapest-insertion heuristic (CIH)

- We begin at any city and find its closest neighbor.
- Then we create a subtour joining those two cities.
- Next, we replace an arc in the subtour (say, arc (i, j) by the combinations of two arcs—(i, k) and (k, j), where k is not in the current subtour—that will increase the length of the subtour by the smallest (or cheapest) amount.
- We continue with this procedure until a tour is obtained.
- After applying this procedure beginning with each city, we take the best tour found.

23

SST: shortest setup Time first

Economic Lot Scheduling
Multiple Items, Single Machine, Arbitrary Schedule

Example : sequence dependent setups

Items	1	2	3	4
Q_i	400	400	500	400
D_i	50	50	60	60
h_i	20	20	30	70
c_i	2000	2500	800	0

Items	1	2	3	4
s_{1k}	-	0.064	0.405	0.075
s_{2k}	0.448	-	0.319	0.529
s_{3k}	0.043	0.234	-	0.107
s_{4k}	0.145	0.148	0.255	-

Optimal Sequence = 1-4-2-3

the same.

if ~~if~~:

if there is a k^*, $s_{ik^*} < s_{ik} + s_{kj}$ for all k.

arc $(i,j) \rightarrow$ arc (i,j,k^*)

In example : $(1,2) \rightarrow (1,4,2) \rightarrow (1,4,2,3)$.

$2 \rightarrow 3 \rightarrow 1 \rightarrow 4$?

∴ it's a cycle.

cost of time:

$(s_{14} + s_{42} + s_{23} + s_{31})$.

Economic Lot Scheduling
Multiple Items, Single Machine, Arbitrary Schedule

Arbitrary Cyclic Schedules

- Sometimes a rotation schedule does not make sense (remember problem with no setup cost)
- For example, we might want to allow a cycle 1,4,2,4,3,4 if item 4 has no setup cost
- No efficient algorithm exists
- If sum of setups > idle time, then the optimal schedule has the property:
  - Each machine is either producing or being setup for production
- An extremely difficult problem with arbitrary setup times

25

Economic Lot Scheduling
Multiple Items, Single Machine, Arbitrary Schedule

Arbitrary Cyclic Schedules

- Feasible solution exists if and only if: $\rho = \sum_i \rho_i < 1$
  (recall that: $x^* = \sum_k s_k / (1 - \sum_i \rho_i)$)
- NP hard problem: no efficient algorithms
- Define sequence as: $j_1, \ldots, j_k, \ldots, j_h$ $(h \geq n)$
  where: j_k=k-th item in the sequence

26

Economic Lot Scheduling
Multiple Items, Single Machine, Arbitrary Schedule

Problem Formulation

- Assume sequence-independent setup
- Formulate as a nonlinear program

$$\min_{\text{sequences}} \min_{\text{lot sizes}} \text{COST}$$

s.t.

demand met over the cycle

demand is met between production runs

27

Economic Lot Scheduling
Multiple Items, Single Machine, Arbitrary Schedule

Notation

- Setup cost and setup times
  $C_{jk} = C_k$, $S_{jk} = S_k$.
- All possible sequences
  $S = \{(j_1, j_2, \ldots, j_h) \;\; h \geq n\}$
- Item k production in l-th position
  $Q^l = Q_{jl} = Q_k$
- Setup time s^l, run time t^l, and idle time u^l may be 0

28

a production time.

Economic Lot Scheduling
Multiple Items, Single Machine, Arbitrary Schedule

Inventory Cost

from the start of the production of item k in the l-th position till the start of the next production of k (in same or next cycle).

- Let x be the cycle time
- Let v be the time between beginning production of k
  生产量 消耗速度 $V = \frac{Q^l t^l}{D^l} = \frac{Q_k t^l}{D_k}$ if $j_l = k$
- Total inventory cost for k is

$$\frac{1}{2} h^l (Q^l - D^l) \left(\frac{Q^l}{D^l}\right) (t^l)^2$$

29

Economic Lot Scheduling
Multiple Items, Single Machine, Arbitrary Schedule

The highest inventory level is: $(Q^l - D^l)t^l$ → total inventory cost for k-th run:

ELSP formulation:

$$\min_{S} \min_{x, t^l, u^l} \frac{1}{x} \left(\sum_{l=1}^{v} \frac{1}{2} h^l (Q^l - D^l) \left(\frac{Q^l}{D^l}\right) (t^l)^2 + \sum_{l=1}^{v} C^l \right)$$

s.t.

$$\sum_{l \in I_k} Q^l = D_k x \qquad k = 1 \ldots n$$

Meet demand for k over cycle x

I_k = all positions where k is produced

L_i = all items produced during v

$$\sum_{l \in L_i} (t^l + s^l + u^l) = \left(\frac{Q^l}{D^l}\right) t^l \qquad l = 1 \ldots h$$

Meet demand k over v (until k produced again)

$$\sum_{l=1}^{h} (t^l + s^l + u^l) = x$$

Cycle length

30

If item k is produced in the l-th position, item k may be produced again within the same cycle.

$$\frac{1}{2} h^l (Q^l - D^l) t^l \cdot V$$

Economic Lot Scheduling
Multiple Items, Single Machine, Arbitrary Schedule

- ELSP master problem: sequence determination
- ELSP subproblem: determination of cycle length, production & idle time
- Frequency Fixing & Scheduling (FFS) heuristic
  3 phases: 1. Computation of relative frequencies y_k, and t_k
  2. Adjustment of relative frequencies → y'_k, and t'_k
  3. Sequencing

Assumption:
the y_k runs of item k are of equal length (t_k), and evenly spaced

31

Economic Lot Scheduling
Multiple Items, Single Machine, Arbitrary Schedule

Optimal cycle, production and idle times subproblem

$$\min_{x,t^i,u^i} \frac{1}{x}\left(\sum_{i=1}^{h}\frac{1}{2}h^i(Q^i-D^i)\left(\frac{Q^i}{D^i}\right)(t^i)^2+\sum_{i=1}^{h}c^i\right)$$

Subject to $$\sum_{j\in L_i}(t^j+s^j+u^j)=\left(\frac{Q^i}{D^i}\right)t^i, \qquad i=1,\ldots,h$$

$$\sum_{j=1}^{n}(t^j+s^j+u^j)=x$$

32

Economic Lot Scheduling
Multiple Items, Single Machine, Arbitrary Schedule

Computing Relative Frequencies

$\rho_k = \frac{D_k}{Q_k}$.

- Let y_k denote the number of times item k is produced in a cycle
- We will
  - simplify the objective function by substituting

$$a_k = \frac{1}{2}h_k(Q_k - D_k)\rho_k = \frac{1}{2}h_k(1-\rho_k)D_k.$$

  - drop the second constraint (meet demand k over V)

$t_k = \frac{\rho_k x}{y_k}$.

33

obj: $\frac{1}{x}\left(\sum_{i=1}^{n}\frac{1}{2}h^i(Q^i-D^i)\left(\frac{Q^i}{D^i}\right)(t^i)^2+\sum_{i=1}^{n}c^i\right)$

$= \frac{1}{x}\left(\sum_{k=1}^{n}\frac{1}{2}y_k h_k(Q_k-D_k)\left(\frac{Q_k}{P_k}\right)(t_k)^2+\sum_{k=1}^{n}y_k c_k\right)$

Economic Lot Scheduling
Multiple Items, Single Machine, Arbitrary Schedule

Computation of relative frequencies y_k, t_k

- Assume sequence that spaces production of item i uniformly over x

$$\min_{y_k,x}\sum_{k=1}^{n}\frac{a_k x}{y_k}+\sum_{k=1}^{n}\frac{c_k y_k}{x}$$

Subject to

$$\sum_{k=1}^{n}\frac{s_k y_k}{x}\le 1-\rho \qquad (\text{from } P_8)$$

34

relax the integer constraints of y_k.
thus relax the constraints on t_k as well.
(basically deleting the first set of constraints).

Economic Lot Scheduling
Multiple Items, Single Machine, Arbitrary Schedule

FFS heuristic, phase 1: frequency fixing

$\min \sum_k \frac{a_k x}{y_k} + \sum_k \frac{c_k y_k}{x} + \lambda\left(\sum_k \frac{s_k y_k}{x} - (1-\rho)\right)$ — Lagrange multiplier

minimization gives: $y_k = x\sqrt{\frac{a_k}{c_k+\lambda s_k}}$ ← Choose "appropriate" x and determine the resulting y_k's

when there is no idle time, λ must satisfy: $\sum_k\left(s_k\sqrt{\frac{a_k}{c_k+\lambda s_k}}\right)=1-\rho$

otherwise: $\lambda=0$ ↓

若 ρ 不为 0, 入 ✱

35

$\min_{y_k,x} \sum_{k=1}^{n}\frac{a_k x}{y_k}+\sum_{k=1}^{n}\frac{c_k y_k}{x}+\lambda\left(\sum_{k=1}^{n}\frac{s_k y_k}{x}-(1-\rho)\right)$

对 x 偏导. partial derivative.

$y_k = x\sqrt{\frac{a_k}{c_k+\lambda s_k}}$.

Economic Lot Scheduling
Multiple Items, Single Machine, Arbitrary Schedule

- Finding λ using Newton's method

let $f_1(\lambda)=\sum_i \frac{s_i\sqrt{a_i}}{\sqrt{c_i+\lambda s_i}}-(1-\rho)$ and $f_1'(\lambda)=\sum_i \frac{-s_i^2\sqrt{a_i}}{2\left[c_i+\lambda s_i\right]^{3/2}}$

thus...

$\lambda_{n+1}=\lambda_n-\frac{f_1(\lambda)}{f_1'(\lambda)}=\lambda_n-\frac{\sum_i \frac{-s_i\sqrt{a_i}}{2\left[c_i+\lambda s_i\right]^{3/2}}}{\sum_i \frac{s_i\sqrt{a_i}}{\sqrt{c_i+\lambda s_i}}-(1-\rho)}, \quad \lambda_n=0$

收敛.

let λ^* = convergent value of λ

if $\lambda^*<0$, set $\lambda=0$

36

$\lambda_{n+1}=\lambda_n-\frac{f(\lambda_n)}{f'(\lambda_n)}$ ☆

↳ $\lambda^*=\lambda_{n+1}=\lambda_n$.

Economic Lot Scheduling
Multiple Items, Single Machine, Arbitrary Schedule

R. Roundy, "Rounding off to powers of 2 in continuous relaxations of capacitated lot sizing problems", Management Science, Vol. 35, No. 12, Dec. 1989, 1432-1441.

- Power-of-2 adjustment for t_i
  - if all items produced in *equal amounts* and at *equal time intervals*, 2^k policies optimal or near optimal.
  - let B = length of base period (to be determined)
    - » *key contribution of paper*
  - Goal : to have 2^k adjustment to production "spacing" be close to x/y_i

$$T_i^{(2)} = B\left(2^k\right) \approx \frac{x}{y_i} = v_i$$

37

Economic Lot Scheduling
Multiple Items, Single Machine, Arbitrary Schedule

FFS heuristic, phase 2: frequency adjustment

- More approximate Power-of-2 adjustment for t_k

$y_k = x\sqrt{\frac{a_k}{C_k + b_k x}} = b_k x$

let $y_{min} = 1 = \min\{b_i\} x = b_{min} x \rightarrow x = \frac{1}{b_{min}}$

Round y_k to closest power-of-2 $\quad y_k = 2^{p_k} \rightarrow \ln[b_k x] = p_k \ln 2 \rightarrow p_k = \frac{\ln[b_k x]}{\ln 2}$.

let $y_k = 2^{p_k} \rightarrow \ln[b_i x] = p_i \ln[2] \rightarrow p_i = \frac{\ln[b_i x]}{\ln[2]}$ (round to nearest integer)

$Q_k t_k y_k = D_k x \Rightarrow t_k = \frac{D_k x}{Q_k y_k} = \frac{D_k x}{y_k}$

38

我认为是为了 y_i 的数值可行.
∵ 整除10的只有2和5. 而5太大
∴ 取 2^n

Economic Lot Scheduling
Multiple Items, Single Machine, Arbitrary Schedule

FFS heuristic, phase 3: sequencing

- Sequencing problem is equivalent to scheduling $\sum y_i$ jobs on y_{max} parallel machines, minimizing the makespan
- Determine $y_{max}=\max(y_1,\ldots y_n)$ = number of parallel machines
- For each item k, there are y_k jobs of length t_k, evenly spaced, i.e.: when m=6, and $y_k=3$, then there are two choices: assign the 3 jobs to (1,3,5) or to (2,4,6)
- Use variant of LPT-heuristic to assign the jobs in decreasing order of y_k, using t_k as tie breaker, also in decreasing order
- additional restriction that jobs must be *evenly spaced*
- *Concatenate* the schedules of the y_{max} machines

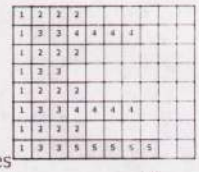
39

y_k 由大到小排, 对 y_k 一样的, 按照 t_k 由大到小排

假装有 y_{max} 个 machine parallel. 排完, 再连起来.

Economic Lot Scheduling
Multiple Items, Single Machine, Arbitrary Schedule

- "Bin packing" solution to production sequencing problem...minimizing the makespan on a set of parallel machines
- *Objective:* space production of each item uniformly over time as possible
- *Example:* suppose $y_1=2^3$ $y_2=y_3=2^2$ $y_4=2^1$ $y_5=2^0$
  - y_1 determines number of bins to pack
  - item 2 loaded before item 3 since $t_2>t_3$
  - if loading of each bin equal... optimal
  - but, adjustment of production times leads to suboptimal results



40

Economic Lot Scheduling
Multiple Items, Single Machine, Arbitrary Schedule

Optimal cycle, production and idle times subproblem ... with established item production in each position

$$\min_{x,t^i,s^i,u^i} \frac{1}{x}\left(\sum_{i=1}^{n}\frac{1}{2}h^i(Q^i - D^i)\left(\frac{Q^i}{D^i}\right)(t^i)^2\right)$$

Subject to $\sum_{j\in L_i}(t^j + s^j + u^j) = \left(\frac{Q^i}{D^i}\right)(t^i) \qquad i = 1,\ldots,h$

$$\sum_{j=1}^{h}(t^j + s^j + u^j) = x$$

substitute in objective function... nonlinear "fractional program" results

41

Economic Lot Scheduling
Multiple Items, Single Machine, Arbitrary Schedule

- Recalculating the t_k and u_k, assuming production of item k is not equally spaced in the cycle
- the item to be produced in each position of L_i is now known

The production times must obey:

$t_i = \frac{g_i}{q} + \sum_{j\in L_i} t^j + s^j + u^j$

in matrix notation:

$\mathbf{t} = (\mathbf{I} - \mathbf{P}^{-1}\mathbf{L})^{-1}\mathbf{P}^{-1}\mathbf{L}(\mathbf{s}+\mathbf{u}) + \mathbf{Y}(\mathbf{s}+\mathbf{u})$

where $\mathbf{P} = diag\left[\frac{g_i}{q}\right]$ and $\mathbf{L} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$

g_i: 是什么 g_i 之和

42

Economic Lot Scheduling
Multiple Items, Single Machine, Arbitrary Schedule

- Item k production times and idle times update...

w_j = row vector j with element $j = 1, \ldots, h$

$= 1$ if $j = k$

$= 0$ otherwise

if $z_k \geq 0$, then $\mathbf{u} = 0$ →

if $z_k < 0$, then $\mathbf{u} = 0$, apply the following adjustment procedure

(1) $\mathbf{t} = \mathbf{Ys}$, initially

(2) let $t^{(2)}$ = production time found using power-of-2 adjusted y_j assigned to position j

if $t < t^{(2)}$, then $t^j \leftarrow t^{(2)}$ for each $j = 1, \ldots, h$

(3) using new $\mathbf{t}$, $\mathbf{u} = \mathbf{Y}^{-1}(\mathbf{t} - \mathbf{Ys})$

43

Economic Lot Scheduling
Multiple Items, Single Machine, Arbitrary Schedule

7.4.1: without setup times, x=2
7.4.2.: with setup times, x=3

1 machine, 4 items:

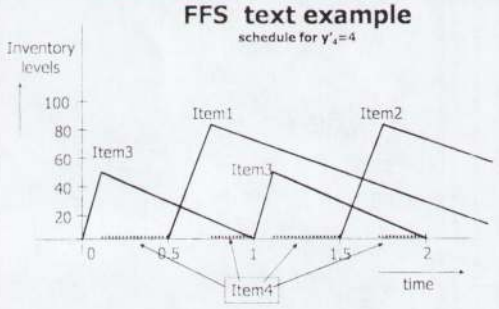
Items	1	2	3	4
Q_j	400	400	500	400
D_j	50	50	60	60
h_j	20	20	30	70
c_j	2000	2500	800	0
s_j	0.5	0.2	0.1	0.2

FFS examples

$y_4 = \infty$ because no cost of setup. can use item 4 fill in the cycle to reduce idle time because of the setup time of others.

Economic Lot Scheduling
Multiple Items, Single Machine, Arbitrary Schedule

FFS text example
schedule for $y'_4 = 4$



45

Economic Lot Scheduling
Multiple Items, Parallel Machines

- Have *m* identical machines in parallel
- Setup cost only
- Item process on only one machine
- *Assume*
  - rotation schedule
  - equal cycle for all machines
- Same as previous multi-item problem
- *Addition:* assignment of items to machines
- *Objective:* balance the load
- *Heuristic:* LPT with $\rho_k = \frac{D_k}{Q_k}$

$$\sum_k \rho_k < m$$

46

Economic Lot Scheduling
Multiple Items, Parallel Machines

Different Cycle Lengths

- *Objective:* assign items to machines to balance the load
- if cycle times are allowed to be different on the machines:
  - calculate cycle time for each item, as if it were a single item model
  - rank items in decreasing order of the cycle times
  - Use sequence in ranking to fill machines as long as $\rho_j < 1$
    NOTE: may lead to infeasible solution! ⇒ swapping
- *Complication:* should not assign items that favor short cycle to the same machine as items that favor long cycle

47

Economic Lot Scheduling
Multiple Items, Flow Shop

- *Model 1*
  - Assume no setup time
  - Assume production rate of each item is identical for every machine
    ⇒ Can be synchronized
  - Reduces to single machine problem $c_k = \sum_{i=1}^{m} c_{ik}$
- *2 machine case*
  - rotation schedules on both machines are the same
  - there may be waiting times on machine 2 (machine 1 may wait)
  - inventory costs remain the same; setup costs double

48

1. use Newton's Method to find α.

2. use $y_k = x\sqrt{\frac{a_k}{c_k + \alpha}}$ to find y_k (non-integer)

   *: $c_4 = 0$. so $y_4 = \infty$.

3. Assume $x = 2$ month ← 选最中间的为 y_k 的参数 接近
   adjust $y_k = 2^{p_k}$.
   use $t_k = \frac{p_k x}{y_k}$ to find all t_k.

4. Use the method on $P_{2,9}$ to sequence.
   not good, change x to 3. again.

5. solve subproblem to determine x, t_k, u_k.

用预测初始 x
计算 y_k, t_k,
从而得到 {frequence sequence.
再用得到的 frequence/sequence
去求最终的 x, t_k, u_k.

↑ with setup times

↓ without setup times

1. find a_j, p_j

2. $y_k = x\sqrt{\frac{a_k}{c_k}}$

3. as above.

4, 5. as above

BTW: $P = \sum_j p_j$

T 157 - 161

☆ in 3, the x is found by calculating without considering the items which can only use once in a cycle.

behavior question

Interval Scheduling, Reservations, and Timetabling

IE 419

Reservation Systems

- Hotel rooms, car rentals, airline tickets (and classroom scheduling)
- You want to have the use of a resource for a given period of time
  - With slack: $p_j < d_j - r_j$
  - Without slack $p_j = d_j - r_j$
- *May not be able to schedule all requests*

在一个window time ($d_j - r_j$) 内安排 job j.

fixed/simply interval scheduling models.

Reservation Systems without Slack

- Reservation system with m machines, n jobs
- Release date r_j, due date d_j, weight w_j
- No slack $p_j = d_j - r_j$
- If processed, job must start at time r_j
- *Should we process the job?*
- *Possible objectives*
  - Maximize $$
  - Maximize resource usage
  - Minimize number of rejected requests
  - Minimize $$ of rejected requests

Reservation Systems without Slack
Example: A Car Rental

- Four types of cars: subcompact, midsize, full size, and sport utility
- Fixed number of each available — Yield management
- Machine = type of car
- Job = customer requesting a car → may be different types
- May request certain machine(s)
  - Job can be processed on a subset of machines, M_j only
  - Midsize substituted for subcompact leads to profit based on machine assignment
  - $w_{ij} = (q_j - c_i) * p_j$
    - q_j is the price charged per day to customer j
    - c_i is the cost (to the rental agency) per day of a car in class i
    - … weights can be job *and* machine dependent

give a larger car at a lower price w_i to keep customer because of no small car.

Reservation Systems without Slack

Feasibility Problem

- Can we process every job?
- Can we assign jobs to machines such that job j is assigned to a set M_j of machines?
- Both relatively easy to solve

Problem Formulation

- General problem is hard
- Special cases are easy
  - All activities have duration of 1 - independent problem for each time slot
  - Decomposition by time unit t and Assignment Problems are solved
  - All weights are 1, all resources in a single set, durations are arbitrary
    - *Maximize the number of scheduled activities*

Reservation Systems without Slack
Problem Formulation

- Integer Programming Problem
- Fixed time periods, t = 1 … H
- Let x_{ij} be 1 if job j assigned to i th machine (and zero otherwise)

$$\max \sum_{i=1}^{m} \sum_{j=1}^{n} w_{ij} x_{ij}$$

$$\sum_{i=1}^{m} x_{ij} \le 1, \quad j = 1, \ldots, n$$ — Every activity is assigned to at most one resource

$$\sum_{j \in J_t} x_{ij} \le 1 \quad i = 1, \ldots, m, \quad t = 1, \ldots, H$$ — Each resource has only one activity per time slot

Jobs requiring processing in period t

Reservation Systems without Slack
Identical Weights and Machines

- Assume $w_j = 1$, arbitrary p_j, and $M_j = \{1,2,\ldots,m\}$
- Does not have a time decomposition
- Simple algorithm maximizes number of jobs that are processed
- Order jobs in increasing release date order

$$r_1 \le r_2 \le \ldots \le r_n$$

7

Reservation Systems without Slack
Algorithm

Step 1
Set $J=\emptyset$ and $j=1$
Step 2
If a machine available at time r_j assign job j to the machine, include it in J and go to Step 4
Step 3
Select j^* such that $C_{j^*} = \max_{k \in J} C_k = \max_{k \in J} \{r_k + p_k\}$

If $C_j = r_j + p_j > C_{j^*}$ do not include j in J and go to Step 4
Else delete job j^* from J, assign j to freed machine & include in J
Step 4
If $j=N$ STOP; otherwise set $j=j+1$ and go back to Step 2

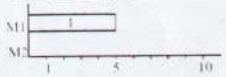
8

Reservation Systems without Slack
Identical Weights and Machines Example

2 machines and 8 jobs

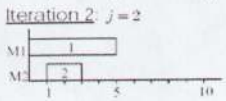
j	1	2	3	4	5	6	7	8
r_j	0	1	1	3	4	5	6	6
d_j	5	3	4	8	6	7	9	8

Iteration 1: $j=1$



J={1}

Iteration 2: $j=2$



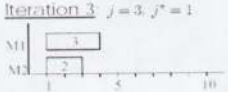
J={1,2}

9

Reservation Systems without Slack
Identical Weights and Machines Example

j	1	2	3	4	5	6	7	8
r_j	0	1	1	3	4	5	6	6
d_j	5	3	4	8	6	7	9	8

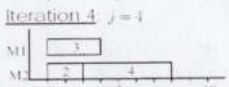
Iteration 3: $j=3$, $j^*=1$



$C_j = r_j + p_j > C_{j^*}$??

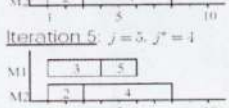
no…J={2,3}

Iteration 4: $j=4$



J={2,3,4}

Iteration 5: $j=5$, $j^*=4$



J={2,3,4,5}

10

Reservation Systems without Slack
Identical Weights and Machines Example

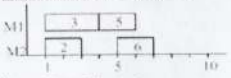
j	1	2	3	4	5	6	7	8
r_j	0	1	1	3	4	5	6	6
d_j	5	3	4	8	6	7	9	8

Iteration 6: $j=6$, $j^*=4$

$C_j = r_j + p_j > C_{j^*}$??

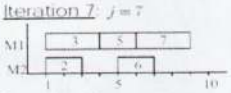
no…J={2,3,5,6}

Iteration 7: $j=7$



J={2,3,5,6,7}

Iteration 8: $j=8$, $j^*=7$



J={2,3,5,6,7,8}

11

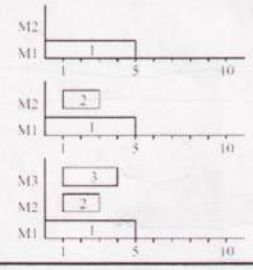
Reservation Systems without Slack
Unlimited Number of Machines

- No slack, arbitrary processing times, equal weights, identical machines
- Infinitely many machines in parallel
- Minimize the number of machines used to process all jobs
- Easily solved by following algorithm:
  - Order jobs by increasing r_j
  - Assign job 1 to machine 1
  - Suppose first j -1 jobs have been processed
  - Try to assign j th job to a machine in use
  - If not possible assign to a new machine
- Graph theory: ***node coloring problem***

12

Reservation Systems without Slack
Unlimited Number of Machines Example

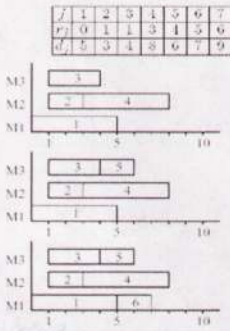
j	1	2	3	4	5	6	7	8
r_j	0	1	1	3	4	5	6	6
d_j	5	3	4	8	6	7	9	8



13

Reservation Systems without Slack
Unlimited Number of Machines Example

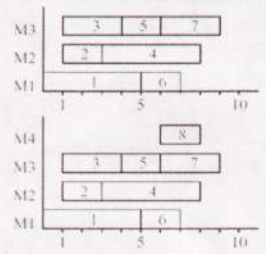
j	1	2	3	4	5	6	7	8
r_j	0	1	1	3	4	5	6	6
d_j	5	3	4	8	6	7	9	8



14

Reservation Systems without Slack
Unlimited Number of Machines Example

j	1	2	3	4	5	6	7	8
r_j	0	1	1	3	4	5	6	6
d_j	5	3	4	8	6	7	9	8



15

Reservation Systems with Slack

- Now allow slack $p_j \le d_j - r_j$
- Trivial case
  - all processing times = 1, identical weights, identical machines
  - decomposition in time possible in this case
- Now assume processing times are equal to some $p \ge 2$
  - Interaction between time intervals
- Barriers algorithm B. Simons, "Multiprocessor Scheduling of Unit-time Jobs with Arbitrary Release Times and Deadlines", SIAM J. of Computing, Vol. 12, No. 2, May 1983
  - wait for critical jobs to be released
    - *otherwise,* start the job with the earliest deadline
  - Slot number l = l th job to start
  - S (l) starting time of l th slot

16

Reservation Systems with Slack
Barrier Algorithm

- Barrier (Slot number, Release time)
  - ordered pair (l ,r)
  - *constraint* : job in slot l can not start before r
- k-partial schedule: jobs inserted in first k slots
- Starting with a k-partial schedule
  - construct a (k +1)-partial schedule, *if possible*
  - *otherwise*, add a new barrier to the barrier list L_b and start over from scratch

17

Reservation Systems with Slack
Earliest Deadline with Barriers

- Selects machine h for job in (k+1)st slot

$$h = \begin{cases} m & k+1 \text{ multiple of } m \\ (k+1) \bmod m & \text{otherwise} \end{cases}$$

取余

- Compute the starting time

$$\tau = \max\{t_1, t_2, t_3, t_4\}$$

18

Reservation Systems with Slack
Computing the Starting Time

- Let τ be the earliest time the job can start

$$\tau = \max\{t_1, t_2, t_3, t_4\}$$

$$t_1 = \max\{S(1), \ldots, S(k)\}$$

Minimum release date of jobs left

$$t_3 = \begin{cases} 0 & k+1 \le m \\ S(k+1-m)+p & \text{otherwise} \end{cases}$$

actually... (k+1)mod(m)

$$t_4 = \max\{r : (k+1, r) \in L_b\}$$

barrier.

19

machine 수 ≥ τ + p.

crisis job → pull job.

→

Reservation Systems with Slack
Selecting a Job

- Select for (k+1)th slot job j ' with the earliest deadline d_j.
- If $d_j \ge \tau + p$, creation of (k+1) partial schedule was successful
- Otherwise a 'crisis' occurred and job j ' is a 'crisis job'
- Crisis routine
  - Backtrack to find job with the latest starting time such that the deadline is greater than the crisis job deadline
  - If no such job, optimal schedule does not include crisis job j'
  - Otherwise this job is a *pull job*, found in slot l_p
  - Add new barrier with the minimum release time r^* of all the jobs J_r after the pull job
  - Start over with (l_p , r^*) added to the barrier list

20

수행.

Reservation Systems with Slack
Barriers Algorithm Example: 2 machines, 4 jobs

Jobs	1	2	3	4
r_j	0	2	5	5
d_j	20	30	19	30

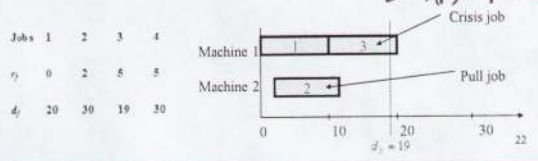
Processing time $p = 10$

21

Reservation Systems with Slack
Barriers Algorithm Example: 2 machines, 4 jobs

- $L_b = \Phi$ max {S(1)=0, S(2)=2} → min (r_j left).
- τ=0 , J_1 has r_1=0 , both machines available , $J_1 \Rightarrow m_1[0,10]$
- τ=r_2=2 , machine 2 available , $J_2 \Rightarrow m_2[2,12]$
- τ=max{t_1, t_2, t_3, t_4}={2, 5, 10, 0}=10
- J_3 and J_4 available , d_3=19<d_4=30 , J_3 selected
- C_3=10+10=20>19 ⇒ J_3 a "crisis job"

S(1)+P=0+10=10



22

Reservation Systems with Slack
Barriers Algorithm Example: 2 machines, 4 jobs

- "Pull job" J_2 has highest slot number with d_j >19
- Restricted set J_r={3} , r_3=5 , L_b={(2,5)}
- Start over...
- τ=0... J_1 has r_1=0 , both machines available , $J_1 \Rightarrow m_1[0,10]$
- τ=max{0, 2, 0, 5} , S(2)=5 on machine 2
- All 3 remaining jobs available at t=5... pick job with earliest $d_j \Rightarrow J_3$
- $J_3 \Rightarrow m_2[5,15]$, C_3=15<d_3=19 , no crisis
- τ=max{5, 5, 10, 0} , S(3)=10 on machine 1
- J_2 and J_4 available , d_2=d_4=30... pick J_2
- τ=10+10=20 < d_2=30 , $J_2 \Rightarrow m_1[10,20]$, no crisis
- τ=max{10, 5, 15, 0} , S(4)=15 on machine 2
- τ=15+10=25 < d_4=30 , $J_4 \Rightarrow m_2[15,25]$, no crisis

(2, 5).

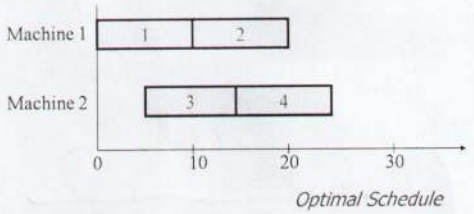
(r, r).

Jobs	1	2	3	4
r_j	0	2	5	5
d_j	20	30	19	30

23

Reservation Systems with Slack
Barriers Algorithm Example: 2 machines, 4 jobs

Second iteration:



Optimal Schedule

24

Reservation Systems with Slack
Generalizations

- Non-identical processing times
  - NP-hard
  - No efficient algorithm exists
  - Need heuristics
- Composite dispatching rule
- Preprocessing: gather information on the flexibility of jobs and machines
  - Let ψ_{ik} be the number of jobs that can be processed on machine i in the slot $[k-1,k]$
  - Let M_j be the set of machines job j can be processed on
- Dispatch least flexible job first on the least flexible machine, etc

25

Reservation Systems with Slack
Priority Indices

- Priority index for jobs

ordered in increasing order of their indices

Low values = high priority

$$I_j = f(w_j / p_j, |M_j|) = \frac{|M_j|}{w_j / p_j} \quad \text{...for example}$$

- Priority index for machines

$$g(\psi_{i,k-1}, \psi_{i,k-2}, \ldots, \psi_{i,k-p_j}) = \begin{cases} \sum_{j=1}^{p_j} \psi_{i,k-j} \Big/ p_j & \text{... or ...} \\ \max\{\psi_{i,k-1}, \psi_{i,k-2}, \ldots, \psi_{i,k-p_j}\} \end{cases}$$

26

$\psi_{i,t}$: # of job can be assigned to machine i during interval $[t-1, t]$.

Max weighted #

Reservation Systems with Slack
Algorithm to Max Weighted Number of Jobs

Step 0
Calculate both priority indices
Order jobs according to job priority index (I_j)

Step 1
Set $j=1$

Step 2
For job $[j]$ select the machine and time slot with lowest rank (available for j)
Discard job $[j]$ if it cannot be processed at all

Step 3
If $j=n$, *STOP*; otherwise set $j=j+1$ and go back to Step 2

27

Reservation Systems with Slack
Algorithm to Max Weighted Number of Jobs

- Example: 4 jobs, 3 machines
- $I(j) = |M_j|^2 / (w_j / p_j)$
- $g(i,k) = \max\{\Psi(i,k+1), \ldots, \Psi(i,k+p_j)\}$
- *Step 0 :*

job j	1	2	3	4
p_j	2	3	4	2
w_j	5	2	5	7
r_j	1	2	3	4
d_j	5	5	8	8
M_j	{1,2,3}	{1,2}	{1,3}	{1}
I(j)	$3^2/(5/2)$ =3.6	$2^2/(2/3)$ =6	$2^2/(5/4)$ =3.2	1/(7/2) =.28

28

Reservation Systems with Slack
Algorithm to Max Weighted Number of Jobs

- Potential job need for machines within job $[r_j, d_j]$

	M_j	1	2	3	4	5	6	7	8
job 1	1,2,3								
job 2	1,2								
job 3	1,3								
job 4	1								
mach 1		0	1	2	3	4	2	2	2
mach 2		0	1	2	2	2	0	0	0
mach 3		0	1	1	2	2	1	1	1

$\psi_{1,t}$ $\psi_{2,t}$ $\psi_{3,t}$

29

Reservation Systems with Slack
Algorithm to Max Weighted Number of Jobs

- *Step 0 (cont.)*

time k	0	1	2	3	4	5	6	7	8
$\Psi(1,k)$		0	1	2	3	4	2	2	2
$\Psi(2,k)$		0	1	2	2	2	0	0	0
$\Psi(3,k)$		0	1	1	2	2	1	1	1

  - job priority index order : 4 - 3 - 1 - 2
- *Step 1 :*
  - pick first job from list : *job 4*

30

Reservation Systems with Slack
Algorithm to Max Weighted Number of Jobs

- *Step 2 :*
  - p_4 = 2 must be accomplished in slots [5,6,7,8]
  - M_4 = {1} , g(1,4) = 4 g(1,5) = 2 **g(1,6)=2**
  - job 4 --> m_1[7,8]
- *Step 3 :*
  - pick job 3 from list
- *Step 2 :*
  - p_3 = 4 must be accomplished in slots [4,5,6,7,8]
  - M_3 = {1,3} , check g(1,3), g(1,4), g(3,3), g(3,4)
    - m_1 excluded because of job 4 conflict
    - g(3,3)=max{2,2,1,1}=2, g(3,4)=max{2,1,1,1}=2 *pick k=4*
    - job 3 --> m_3[5,6,7,8]

time k	0	1	2	3	4	5	6	7	8
Ψ(1,k)		0	1	2	3	4	2	2	2
Ψ(2,k)		0	1	2	2	2	0	0	0
Ψ(3,k)		0	1	1	2	2	1	1	1

Reservation Systems with Slack
Algorithm to Max Weighted Number of Jobs

- *Step 3 :*
  - pick job 1 from list
- *Step 2 :*
  - p_1 = 2 must be accomplished in slots [2,3,4,5]
  - M_1 = {1,2,3} , check g(i,1), g(i,2), g(i,3) for i = 1,2 and g(3,1), g(3,2)
  - g(i,k):

k	1	2	3
m_1	2	3	4
m_2	2	2	2
m_3	1	2	----

pick m_3 and k=1

  - job 1 --> m_3[2,3]

time k	0	1	2	3	4	5	6	7	8
Ψ(1,k)		0	1	2	3	4	2	2	2
Ψ(2,k)		0	1	2	2	2	0	0	0
Ψ(3,k)		0	1	1	2	2	1	1	1

32

Reservation Systems with Slack
Algorithm to Max Weighted Number of Jobs

- *Step 3 :*
  - pick job 2 from list
- *Step 2 :*
  - p_2 = 3 must be accomplished in slots [3,4,5]
  - M_2 = {1,2} , check g(1,2)=4, **g(2,2)=2**
  - job 2 → m_2[3,4,5] *...but m_1[3,4,5] eliminates need for m_2*

time k	0	1	2	3	4	5	6	7	8
Ψ(1,k)		0	1	2	3	4	2	2	2
Ψ(2,k)		0	1	2	2	2	0	0	0
Ψ(3,k)		0	1	1	2	2	1	1	1

slot	1	2	3	4	5	6	7	8
m_1							4	4
m_2			2	2	2			
m_3		1	1		3	3	3	3

33

Timetabling

- Infinite identical machines in parallel
- All of *n* jobs must be processed
- Tooling constraints
  - Many tools
  - Need one or more tools for each job
- Resource constraints
  - Single resource of quantity *R*
  - Need certain amount for each job

34

Timetabling
Resource Constraints

- One type of tool but *R* units of it (resource)
- Job *j* needs R_j units of this resource
- Clearly, if $R_j + R_k > R$ then job *j* and *k* cannot be processed at the same time, etc
- Applications
  - scheduling a construction project (*R*=crew size)
  - exam scheduling (*R*=number of seats)
- Workforce capacity
  - n activities
  - An infinite number of resources
  - Each activity requires W_j workers
  - You only have W workers
  - Find a schedule that minimizes makespan

35

Timetabling
Resource Constraints

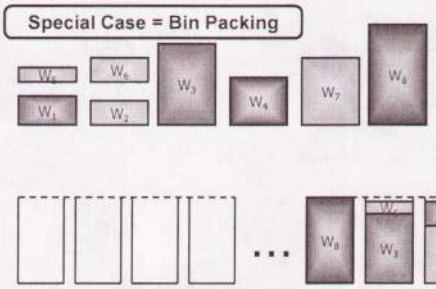
Special Case: Exam Scheduling

- All exams have the same duration
- An exam room has capacity W
- Course j has W_j students
- All students in course j must write the exam at the same time
- Find a timetable for all n exams in the minimum amount of time

36

Timetabling
Resource Constraints

Special Case = Bin Packing



Pack the objects into the bins to minimize the number of bins that are used

Each bin has capacity = W

37

Timetabling
Solving the Bin-Packing Problem

- First fit (FF) heuristic
  - Always put an item in the first bin it fits into
  - Worst case bound: $C_{max}(FF) \le \frac{17}{10} C_{max}(OPT) + 2$
- Example: Assume 18 items and R=2100
  - Jobs 1-6 require 301 resources
  - Jobs 7-12 require 701 resources
  - Jobs 13-18 require 1051 resources
- FF heuristic:
  - We assign the first 6 jobs to one interval (301·6=1806) ≤2100
  - We then assign jobs two at a time to next 3 intervals (701·2=1402) ≤2100
  - We then assign just one of the remaining jobs to each interval
- Very poor performance due to *initial order of jobs*

38

min ~~# of A~~ . Cmax .

Timetabling
Solving the Bin-Packing Problem

First Fit Decreasing (FFD) FFD

- An improvement of FF:
  Order jobs in decreasing order of R_j first
- Worst case bound

$$C_{max}(FFD) \le \frac{11}{9} C_{max}(OPT) + 4$$

- FF and FFD can be extended to arbitrary release dates

39

Timetabling
Solving the Bin-Packing Problem

activities	1...6	7...12	13...18
W_j	301	701	1051

- First Fit Decreasing Example:
  - order jobs of FF example in decreasing R_j order
    - Job List: 13,...,18,7,...,12,1,...,6 W = 2100

check through the order .

bin 1	13[1051]	14[1051]	7[701]	8[701]	1[301]	2[301]
bin 2	14[1051]	8[701]	2[301]			
bin 3	15[1051]	9[701]	3[301]			
bin 4	16[1051]	10[701]	4[301]			
bin 5	17[1051]	11[701]	5[301]		optimal	
bin 6	18[1051]	12[701]	6[301]		solution	

40

different tooling constraints || min Cmax .

Timetabling
Operator or Tooling Constraints

People are Different...

- So far we have been assuming that we have W people that are all equivalent
  - Any set of W_j people could do activity j
- Obviously this isn't true in many applications
  - E.g., Medical or language specialties
- The operator has to have specific skills to do the activity
  - Able to speak French, certified accountant
- A specific tool is needed
  - 1/8" drill bit, refrigerator truck
- If two activities require the same operator or tool, they cannot be done at the same time

41

Timetabling
Tooling Constraints

- One of each tool
- *Objective:* minimize makespan
- Special case of resource-constrained project scheduling problem with one of each resource, that is, R_i = 1, and no precedence constraints
- NP-hard
- Assume all processing times = 1 ... no efficient algorithms exist
- *Equivalent to node coloring problem*

42

Timetabling
Tooling Constraints

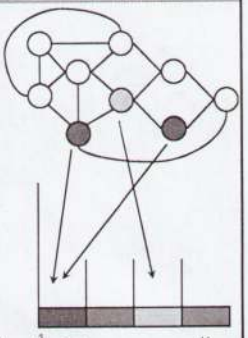
- **Node Coloring Problem**
  - Job = node
  - Jobs need same tool = arc between nodes
  - Feasibility problem:
    - Can the graph be colored with H colors (timeslots) ?
  - Optimization problem:
    - What is the lowest number of colors needed?
    - *Chromatic number* of the graph

43

Timetabling
Tooling Constraints

Mapping from Graph Coloring to Timetabling

- Nodes are activities
- Arcs mean the activities require the same resource
- Colors are time slots
  - Minimizing the number of colors is minimizing makespan



one color on time slots

44

Timetabling
Tooling Constraints

- Closely related to reservation problem with zero slack and arbitrary processing times
- Special case of timetabling problem
  - tools in common = overlapping time slots
  - tools in common ⇒ nodes connected
  - colors = machines
  - minimizing colors = minimizing machines
  - adjacent time slots in reservation problem *vs* tools need not be adjacent in timetabling problem (thus, a *harder* problem)

45

Reservation Systems without Slack
Node Coloring Problem

- Consider a graph with n nodes
- If an arc (j,k) connects nodes j and k , they cannot be colored with the same color
  - jobs which overlap have to be on different machines
- How many colors do we need to color the graph?

Number of colors needed = Number of machines needed

- Node coloring characterization provides linkage to *Timetabling Problem with Tool Constraints*

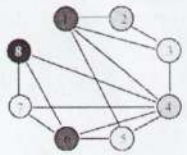
46

Reservation Systems without Slack
Node Coloring Problem

Example formulated as Node Coloring Problem

j	1	2	3	4	5	6	7	8
r_j	0	1	1	3	1	5	6	6
d_j	5	3	4	8	6	7	9	8

corresponding graph coloring problem:



47

Variable and Value Ordering Heuristics

- Many heuristics exist for graph coloring
  - degree of node = number of arcs connected to node
  - saturation level = number of colors connected to node
- *Intuition:*
  - Color high degree nodes first
  - Color high saturation level nodes first
- How do you pick a node (variable) to color next?
  - Smallest domain first
  - Maximum forward degree
- How do you pick a color (value) for that node?
  - Randomly
  - Lexicographically
    - Order colors and assign the lowest possible

48

Timetabling
Graph Coloring Heuristic

Step 1
Order nodes in decreasing order of degree

Step 2
Use color 1 for first node

Step 3
Choose uncolored node with maximum saturation level, breaking ties according to degree

Step 4
Color using the lowest possible number color

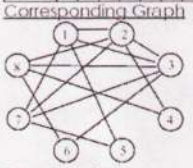
Step 5
If all nodes colored, STOP; otherwise go back to Step 3

49

Timetabling
Graph Coloring Heuristic Example

Jobs	1	2	3	4	5	6	7	8
	1	1	1	1	1	1	1	1
Tool 1	1	1	1	0	0	0	1	0
Tool 2	0	1	0	1	0	0	0	1
Tool 3	1	0	0	0	1	0	1	0
Tool 4	0	0	1	0	0	1	0	1
Tool 5	0	0	0	1	0	0	0	0

Corresponding Graph



Preprocessing:

Jobs(nodes)	1	2	3	4	5	6	7	8
degree	4	5	5	2	2	2	4	4

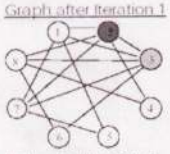
50

Timetabling
Graph Coloring Heuristic Example

Initial: choose node 2 and color it red (color 1)

Iteration 1: all neighbors of 2 (1,3,4, 7,8) have saturation level 1;
3 has highest degree
color node 3 green (color 2)

Graph after Iteration 1



Data after Iteration 1:

Jobs(nodes)	1	2	3	4	5	6	7	8
saturation level	2	-	-	1	0	1	2	2
degree	2	-	-	1	2	1	2	2

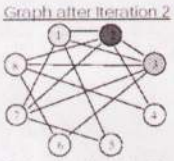
connections to uncolored nodes

51

Timetabling
Graph Coloring Heuristic Example

Iteration 2: 1,7,8 have max.sat. level;
all have degree 2;
choose node 1 and color it yellow (color 3)

Graph after Iteration 2



Data after Iteration 2:

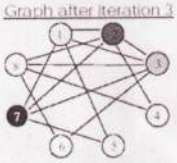
Jobs(nodes)	1	2	3	4	5	6	7	8
saturation level	-	-	-	1	1	1	3	2
degree	-	-	-	1	1	1	1	2

52

Timetabling
Graph Coloring Heuristic Example

Iteration 3: 7 has max.sat. level;
color node 7 blue (color 4)

Graph after Iteration 3



Data after Iteration 3:

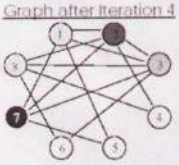
Jobs(nodes)	1	2	3	4	5	6	7	8
saturation level	-	-	-	1	2	1	-	2
degree	-	-	-	1	0	1	-	2

53

Timetabling
Graph Coloring Heuristic Example

Iteration 4: 5,8 have max.sat. level;
8 has highest degree
color node 8 yellow (color 3)

Graph after Iteration 4



Data after Iteration 4:

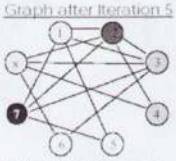
Jobs(nodes)	1	2	3	4	5	6	7	8
saturation level	-	-	-	2	2	2	-	-
degree	-	-	-	0	0	0	-	-

54

Timetabling
Graph Coloring Heuristic Example

Iteration 5: 4,5,6 have max sat. level;
all have degree 0;
choose node 4 and color it green
(color 2)

Graph after Iteration 5



Data after Iteration 5:

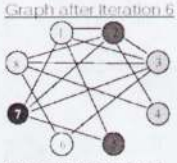
Jobs(nodes)	1	2	3	4	5	6	7	8
saturation level	-	-	-	-	2	2	-	-
degree	-	-	-	-	0	0	-	-

55

Timetabling
Graph Coloring Heuristic Example

Iteration 6: 5,6 have max sat. level;
both have degree 0;
choose node 5 and color it red
(color 1)

Graph after Iteration 6



Data after Iteration 6:

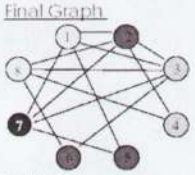
Jobs(nodes)	1	2	3	4	5	6	7	8
saturation level	-	-	-	-	-	2	-	-
degree	-	-	-	-	-	0	-	-

56

Timetabling
Graph Coloring Heuristic Example

Iteration 7: only node 6 is left;
color node 6 red (color 1)

Final Graph



Solution:

jobs 2, 5, and 6 at time 1
jobs 3 and 4 at time 2
jobs 1 and 8 at time 3
job 7 at time 4

57

Timetabling
Graph Coloring Heuristic Example

Relation to Interval Scheduling

Remark: For the given example the tools can not be ordered such that for all jobs the used tools are adjacent (i.e. the resulting graph is not an interval graph). Thus the instance can not be seen as an interval scheduling instance.

Change of the data:
assume job 2 needs besides tool 1 and 2 also tool 4

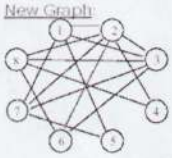
New data:

Jobs	1	2	3	4	5	6	7	8
p_j	1	1	1	1	1	1	1	1
Tool 1	1	1	1	0	0	0	1	0
Tool 2	0	1	0	1	0	0	0	1
Tool 3	1	0	0	0	1	0	1	0
Tool 4	0	1	1	0	0	1	0	1
Tool 5	0	0	0	1	0	0	0	0

58

Timetabling
Graph Coloring Heuristic Example

New Graph:



Tool Renumbering:

Jobs	1	2	3	4	5	6	7	8
p_j	1	1	1	1	1	1	1	1
Tool 3	1	0	0	0	1	0	1	0
Tool 1	1	1	1	0	0	0	1	0
Tool 4	0	1	1	0	0	1	0	1
Tool 2	0	1	0	1	0	0	0	1
Tool 5	0	0	0	1	0	0	0	0

59

Timetabling
Graph Coloring Heuristic Example

Transform

slot 1: time 1 → tool 3
slot 2: time 2 → tool 1
3: time 3 → tool 4
4: time 4 → tool 2
time 5 → tool 5

Jobs	1	2	3	4	5	6	7	8
p_j	1	1	1	1	1	1	1	1
Tool 3	1	0	0	0	1	0	1	0
Tool 1	1	1	1	0	0	0	1	0
Tool 4	0	1	1	0	0	1	0	1
Tool 2	0	1	0	1	0	0	0	1
Tool 5	0	0	0	1	0	0	0	0

decreasing order.

Resulting Interval Scheduling Prob.:

Job	1	2	3	4	5	6	7	8
r_j	0	1	1	3	0	2	0	2
d_j	2	4	3	5	1	3	2	4
p_j	2	3	2	2	1	1	2	2

P29

without slack.

正好等到1都连在一起 不可以选择类比.

60

Timetabling
Problem Equivalence Example

► Suppose now we have a reservation system:

Job	1	2	3
p_j	2	3	1
r_j	0	2	3
d_j	2	5	4

61

Timetabling
Equivalent Timetabling Problem

	Job	1	2	3
Job 1 must be processed in time [0,2]	Tool 1	1	0	0
	Tool 2	1	0	0
Job 2 must be processed in time [2,5]	Tool 3	0	1	0
	Tool 4	0	1	1
	Tool 5	0	1	0

62

Timetabling
Scheduling Meetings Example

► Schedule 4 people to attend a subset of 5 meetings
- meetings = jobs and people = tools
- all meetings are one hour

	1	2	3	4	5
Joe	1	1	0	1	1
Lisa	1	1	1	0	0
Jane	1	0	1	0	0
Larry	0	1	0	1	1

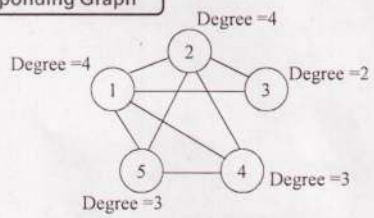
overlap. 連續

eg. 4 – Joe → 1,2,5 } ⇒ 4 – 1,2,5.
4 – Larry – 2,5. }

63

Timetabling
Scheduling Meetings Example

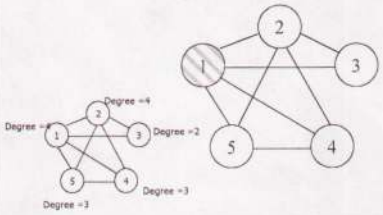
Corresponding Graph



64

Timetabling
Scheduling Meetings Example

Can select either job 1 or 2 first
Say, select 1 and color with the first color



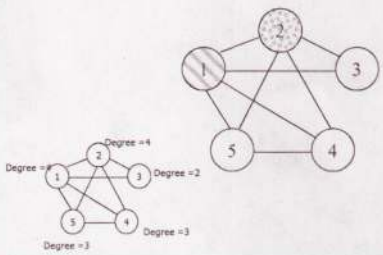
Saturation level = 1 for every node

⇒ Select 2 because highest degree

65

Timetabling
Scheduling Meetings Example

Color Job 2 with the next color



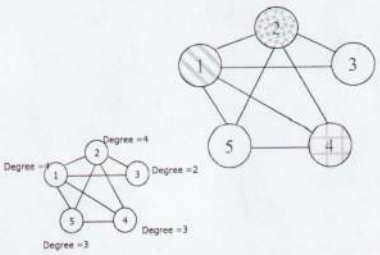
Saturation level = 2 for every node

⇒ Select 4 because highest degree

66

Timetabling
Scheduling Meetings Example

Color Job 4 with the next color



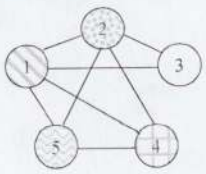
Saturation level =
2 for node 3
3 for node 5

⇒ Select 5 next

67

Timetabling
Scheduling Meetings Example

Color Job 5 with the next color

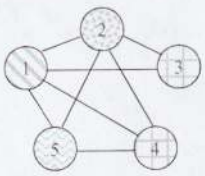


⇒ Select 3 next

68

Timetabling
Scheduling Meetings Example

Color Job 3 with same color as Job 4



Had to use 4 colors ⇒ Makespan = 4

69