More Advanced Single Machine Models

Total Earliness And Tardiness

- Non-regular performance measures $\sum E_j + \sum T_j$
- Early jobs (Set j_1) and Late jobs (Set j_2) are scheduled according to LPT and SPT.
- **Minimizing Total Earliness And Tardiness with a loose due date.**

Assume:

1. $d_j = d$.
2. $p_1 \geq p_2 \geq p_3 \geq \ldots\ldots\ldots\ldots\ldots\ldots \geq p_n$

Step 1: Assign job 1 to set j_1.

set k=2

Step 2: Assign job k to set j_1 and job k+1 to set j_2 or vice versa.

Step 3: If $k+2 \leq n - 1$, set k=k+2 and go to step 2.

If $k+2 = n$, assign job n to either j_1 or j_2 and STOP.

If $k+2 = n + 1$, all jobs have been assigned ; STOP.

- Flexible in assigning jobs to sets j_1 and j_2..
- Assignment is such that the total processing times of set j_1 is minimized.

Total Earliness And Tardiness (Cont.)

Assume:

1. d_j=d.
2. $p_1 \geq p_2 \geq p_3 \geq \ldots\ldots\ldots\ldots\ldots \geq p_n$

- **Minimizing Total Earliness And Tardiness with a tight due date.**

Step 1: Set τ_1 =d and $\tau_2 = \sum p_j - d$
Set k=1

Step 2: If $\tau_1 > \tau_2$, assign job k to the first unfilled position in the sequence and set $\tau_1 = \tau_1 - p_k$.
If $\tau_1 < \tau_2$, assign job k to the last unfilled position in the sequence and set $\tau_2 = \tau_2 - p_k$.

Step 3: If k < n , set k = k+1 and go to step 2.
If k = n , STOP.

Example:

Jobs	1	2	3	4	5	6
p_j	106	100	96	22	20	2

τ_1	τ_2	Sequence
180	166	1XXXXX
74	166	1XXXX2
74	66	13XXX2
-22	66	13XX42
-22	44	13X542
-22	12	136542

Total Earliness And Tardiness (Cont.)

- If we consider $\sum w'Ej + \sum w''Tj$,where the weights are not necessary the same for the two performance measures but the due dates are same, the earlier algorithms can be generalized easily for solving this problem.
- Now if we consider $\sum w_j'Ej + \sum w_j''Tj$ and $d_j=d$, then the weighted LPT and weighted SPT rules have to be used for sequencing.
- Now if we consider $\sum w_j'Ej + \sum w_j''Tj$ and $d_j \neq d$, the problem is NP hard.
- Due to different due dates it might not be optimal to process the jobs without interruption. Idle times in between consecutive jobs might be necessary.
- Given a predetermined ordering of the jobs, the timings of the processing of the jobs and the idle times can be computed in polynomial times.

- **Lemma 1**: If $d_{j+1} – d_j \leq p_{j+1}$, then there is no idle time between jobs $_j$ and $_{j+1}$.

Three cases:

1. J is early.
2. J is completed exactly at its due date.
3. J is late.

- **Lemma 2**: In each cluster in a schedule, the early jobs proceed the tardy job. Moreover, if the jobs j and j+1 are in the same cluster and are both early, then $E_j \geq E_{j+1}$. If the jobs are both late ,then $T_j \leq T_{j+1}$.

For a cluster;

$$d_{j+1} – d_j \leq p_{j+1}.$$

Subtracting $t+p_j$ from both sides, we get

$$d_{j+1} - d_j - t - p_j \leq p_{j+1} - t - p_j.$$

Solving we get,

$$d_j - C_j \geq d_{j+1} - C_{j+1}$$

- The job sequence 1,2,3…….n can be decomposed into m clusters with each cluster representing a subsequence.
- We compute the optimal shift for each cluster.
- For a cluster with jobs k,k+1,…..,l; let

$\Delta(j) = \sum w'_l + \sum w''_l \qquad l = k \text{ to } j$

- A block is a sequence of clusters that are processed without interruption.
- Let $E(r) = E_{jr} = d_{jr} - C_{jr}$ where j_r is the last job in cluster σ_r that is early.
- Hence $E(r) = \min_j (d_{jr} - C_{jr})$; where $k \le j \le j_r$.
- Now let $\Delta(r) = \Delta j_r = \max \Delta(j)$; where $k \le j \le j_r$.
- If none of the jobs in the cluster is early, then $E(r) = \infty$ and $\Delta(r) = - \sum w''_l$.
- If $E(r) \ge 1$ for the last early job in every cluster of the block, a shift of the entire block by one unit time to the right decreases the total cost by $\sum \Delta(r)$ (the summation is over the block).

Optimizing timings given a predetermined sequence

- **Algorithm:**

Step1: Identify the clusters and compute $\Delta(r)$ and $E(r)$ for each cluster.

Step2 : Find the smallest s s.t. $\sum \Delta(r) \leq 0$.

Set the original C_k for each job of the first s cluster.

If s = m, then STOP; other wise go to step 3.

If no such s exists, then go to step 4.

Step3: Remove the first s clusters from the list.

Go to step 2 to consider the reduced sets of cluster.

Step 4: Find minimum (E(1)……E(m)).

Increase all C_k by minimum (E(1)……E(m)).

Eliminate all early jobs that are no longer early.

Update $E(r)$ and $\Delta(r)$. Go to step 2.

Optimizing Timings Given A Predetermined Sequence

Jobs	1	2	3	4	5	6	7
pj	3	2	7	3	6	2	8
dj	12	4	26	18	16	25	30
w_{j1}	10	20	18	9	10	16	11
w_{j2}	12	25	38	12	12	18	15

- $\sigma_1 = 1,2$; $\sigma_2 = 3,4,5$; $\sigma_3 = 6,7$
- Completion times will be 3,5,12,15……..

$E(r) = \text{Min}(d_j - c_{j)}$ and $\Delta(r) = \max \Delta_j$

Cluster	1	2	3
E(r)	9	3	2
Δ(r)	-15	15	1

Cluster	2	3
E(r)	1	Infinity
Δ(r)	15	-33

The optimal completion times are:

3,5,14,17,23,25,33

Primary and Secondary Objectives

- $\alpha \mid \beta \mid \gamma_1$(Opt.), γ_2.
- Lemma: For the single machine problem with n jobs subject to the constraint that all due dates have to be met, there exists a schedule that minimizes $\sum C_j$ in which job k is scheduled last, if and only if
  1. $d_k \geq \sum p_j$
  2. $p_k \geq p_L$, for all L such that $d_L \geq \sum p_j$

- Minimizing total completion times with deadlines (backward algorithm).
- **Algorithm:**

Step 1: Set $k = n$, $\tau = \sum p_j$, $j^c = \{1,2,\ldots\ldots,n\}$

Step 2: Find k^* in j^c s.t. $d_{k^*} \geq \tau$ and $p_{k^*} \geq p_L$, for all jobs L in j^c s.t. $d_L \geq \tau$

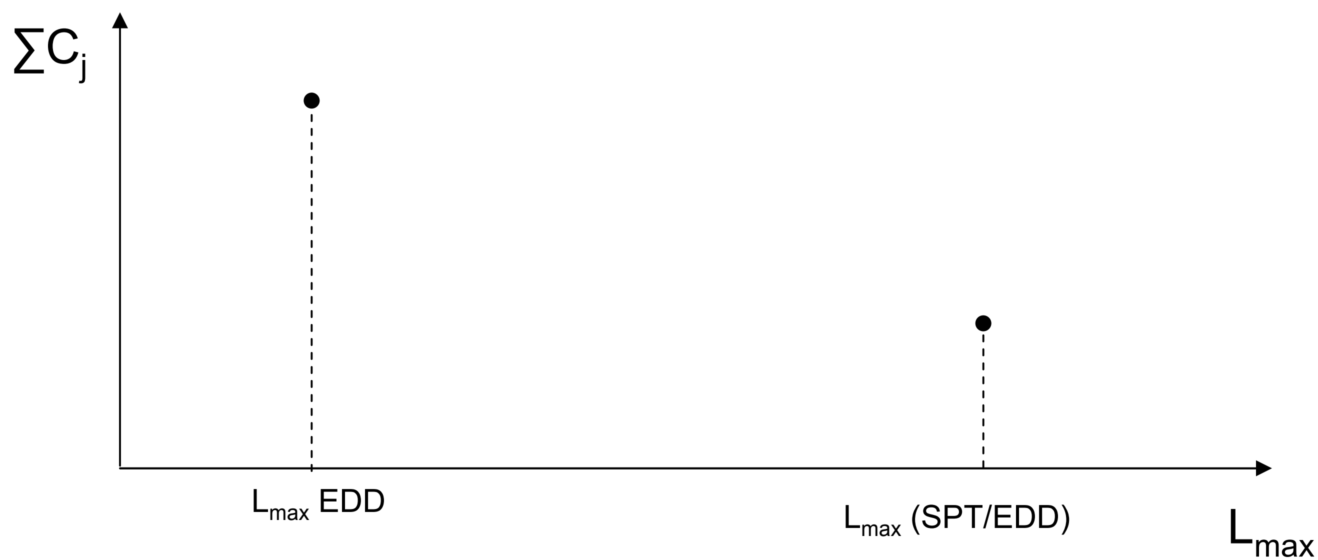
Put job k^* in position k of the sequence.

Step 3: Decrease k by 1 ; decrease τ by p_{k^*}. Delete job k^* from j^c.

Step 4: If $k \geq 1$ go to Step 2, otherwise STOP.

- **Pareto-optimal schedule**: is the one in which it is not possible to decrease the value of one objective without increasing the value of the other.

  $1 | \beta | \theta_1\gamma_1 + \theta_2\gamma_2$; where θ_1, θ_2 are the weights of the two objectives.



Trade-off between total completion time and maximum lateness

SEQUENCE-DEPENDENT SETUP PROBLEMS

Sequence-Dependent Setup Problems

1. An algorithm which gives an optimal schedule
   with the minimum makespan with sequence-dependent setup times
   $1 \mid S_{jk} \mid C_{max}$

Single machine: r_j=0, no sequence dependent setup times $\Rightarrow C_{\max} = \sum_j p_j$

$1 \mid S_{jk} \mid C_{max}$ **NP hard**

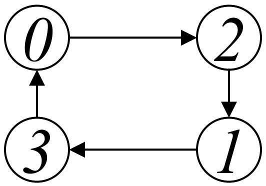
- Set-up times have a special structure and hence an efficient solution procedure is possible.
- Consider a structure where two parameters associated with job j : a_j *and* b_j

1. At the completion of the job the machine state is b_j
2. To start the job the machine must be in state a_j

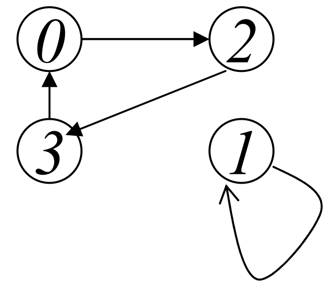
- $s_{jk} = | a_k - b_j |$ is the total setup time necessary to bring the machine from state b_j to a_k state.
- Machine speed.
- Travelling Salesman Problem with $n+1$ cities $j_0, j_1, \ldots, j_n$. The additional city C_o has parameters a_o & b_o.

$k = \phi(j)$ is the relation that maps each element of $\{0, 1, 2, \ldots, n\}$ onto a unique element of $\{0, 1, 2, \ldots, n\}$ .Traveling salesman is leaving city j for city k.

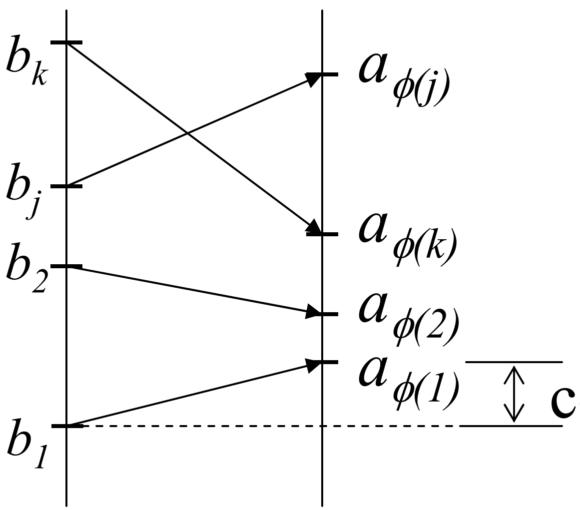
$$\{0, 1, 2, 3\} \rightarrow \{2, 3, 1, 0\}$$



$$\phi(0) = 2$$
$$\phi(1) = 3$$
$$\phi(2) = 1$$
$$\phi(3) = 0$$



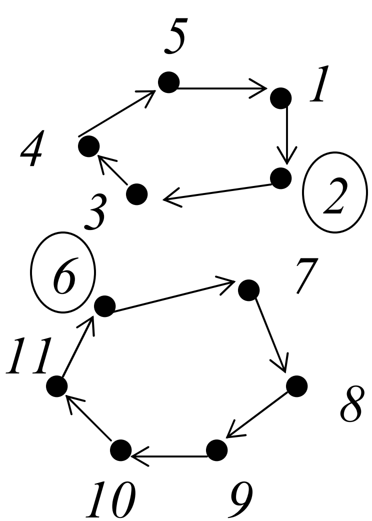
$$\{0, 1, 2, 3\} \rightarrow \{2, 1, 3, 0\}$$

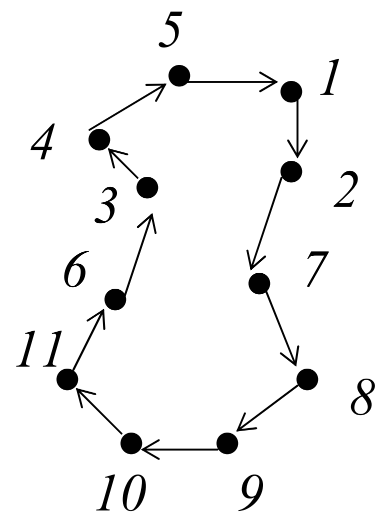


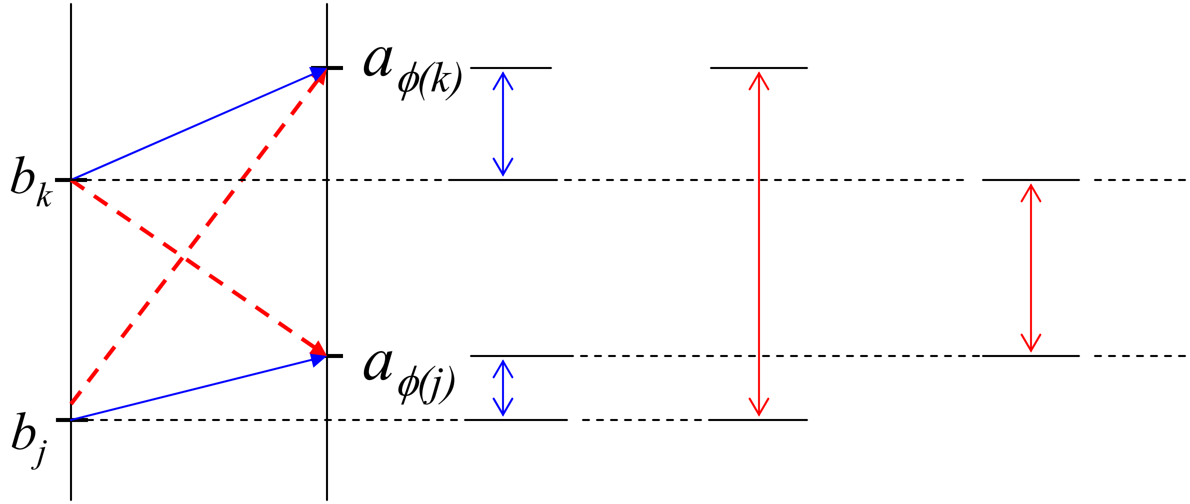
cost of going from 1 to $\phi(1)$ is $| a_{\phi(1)} - b_1 |$

Swap $I(j,k)$ applied to a permutation ϕ produces another permutation ϕ' by affecting only the assignments of j and k and leaving the others unchanged.

$\phi'(k) = \phi(j)$
$\phi'(j) = \phi(k)$
$\phi'(l) = \phi(l), \quad l \neq j , k$







change in cost due to swap $I(j, k)$

Lemma. If the swap causes two arrows that did nor cross earlier

{|[bj, bk] ∩ [a φ , bφ]|}.

to cross, then the cost of the tour $C_\phi I(j,k)$ increases and vice versa.

$$C\phi\, I(j,k) = \left\| [\, b_j, b_k \,] \cap [a_{\phi(j)}, b_{\phi(k)}] \right\|.$$

Here, $\| [\, a,b \,] \| = \begin{cases} 2\,(b-a) & \text{if } b \geq a \\ 2\,(a-b) & \text{if } b < a \end{cases}$

- **Lemma**. An optimal permutation mapping ϕ^* is obtained if :

  bj ≤ bk implies that $a_{\phi(j)} \leq a_{\phi(k)}$.
- This is an optimal permutation mapping and not necessary a feasible tour.
- ϕ^* might consist p distinct sub tours.
- A swap on i & j, belonging to different sub-tours, will cause them to cross each other and thus coalesce into one and increase the cost.
- Hence we select the cheapest arc that connects two of the p sub-tours and so on.

- **Lemma.** The collection of arcs that connect the undirected graph with the least cost contain only arcs that connect city j to city j+1.

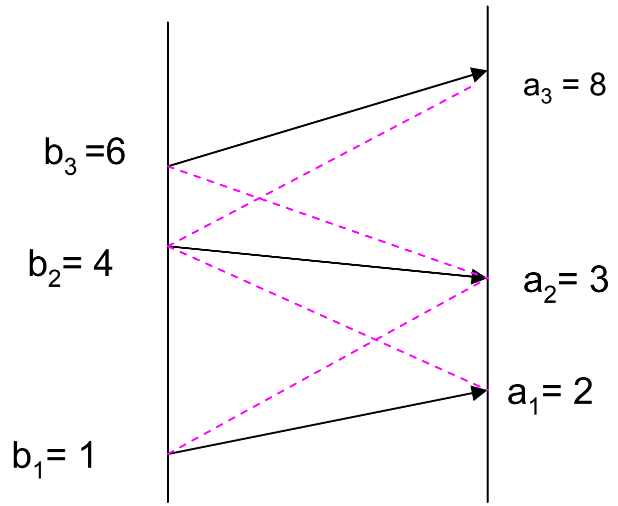
Consider $k > j+1$.

$$C\phi I(j,k) = \| [b_j, b_k] \cap [a_{\phi(j)}, b_{\phi(k)}] \|$$
$$\geq \sum_i \| [b_i, b_{i+1}] \cap [a_{\phi^*(i)}, b_{\phi^*(i+1)}] \|$$

for i= j,……., k-1

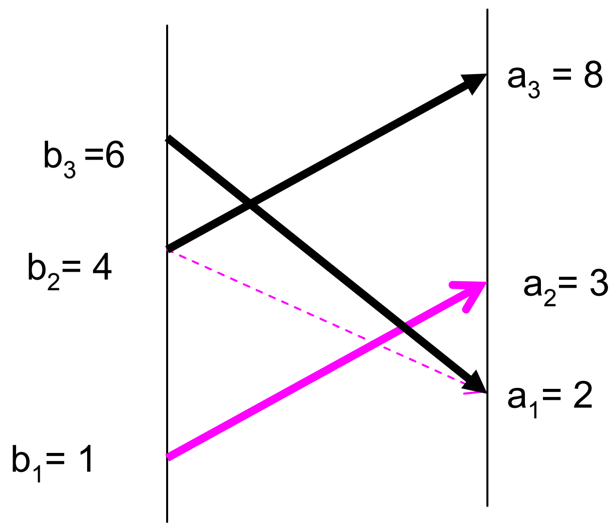
Hence the cost of swapping two nonadjacent arrows is at least equal to the cost of swapping all arrows between them.

- Here no arrows are allowed to cross. But in order to connect two sub-tours this condition might not be valid.



$C_\phi\, I(1,2) = \left\| [\ 1,4\ ] \cap [\ 2,3\ ] \right\| = 2(3\text{-}2) = 2$

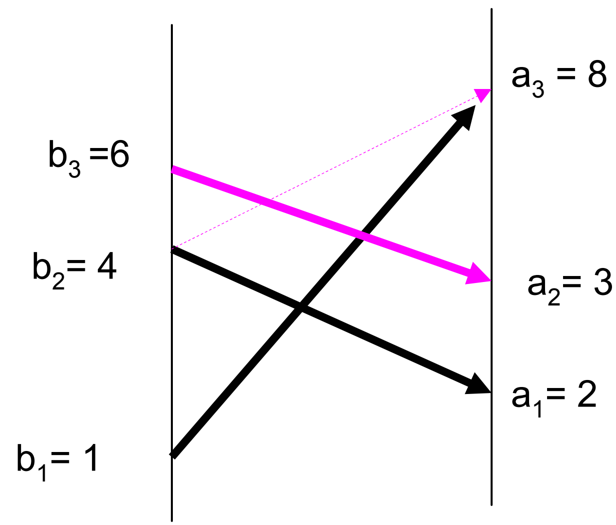
$C_\phi\, I(2,3) = \left\| [4,6\ ] \cap [\ 3,8\ ] \right\| = 2(6\text{-}4) = 4$



I(1,2) then I(2,3)

$C\phi\ I(1,2) = \left\| [1,4\ ] \cap [\ 2,3\ ] \right\| = 2(3\text{-}2) = 2$

$C\phi\ I(2,3) = \left\| [4,6\ ] \cap [\ 2,8\ ] \right\| = 2(6\text{-}4) = 4$



I(2,3) then I(1,2)

$C\phi\ I(2,3) = \left\| [4,6\ ] \cap [\ 3,8\ ] \right\| = 2(6\text{-}4) = 4$

$C\phi\ I(1,2) = \left\| [1,4\ ] \cap [\ 2,8\ ] \right\| = 2(4\text{-}2) = 4$

Here cost increased.

A node is of ***Type 1*** if $b_j \leq a_{\phi(j)}$ (arrow points up)
A node is of ***Type 2*** if $b_j > a_{\phi(j)}$ (arrow points down)

A swap is of ***Type 1*** if lower node is of ***Type 1***
A swap is of ***Type 2*** if lower node is of ***Type 2***

If swaps $I(j, j+1)$ of ***Type 1*** are performed in decreasing order of the node indices,
followed by swaps of ***Type 2*** in increasing order of the node indices
then a single tour is obtained without changing any $C_{\phi^*} I(j, j+1)$ involved in the swaps

Algorithm + Example

7 jobs

b_j	1	15	26	40	3	19	31
a_j	7	16	22	18	4	45	34

Step 1.

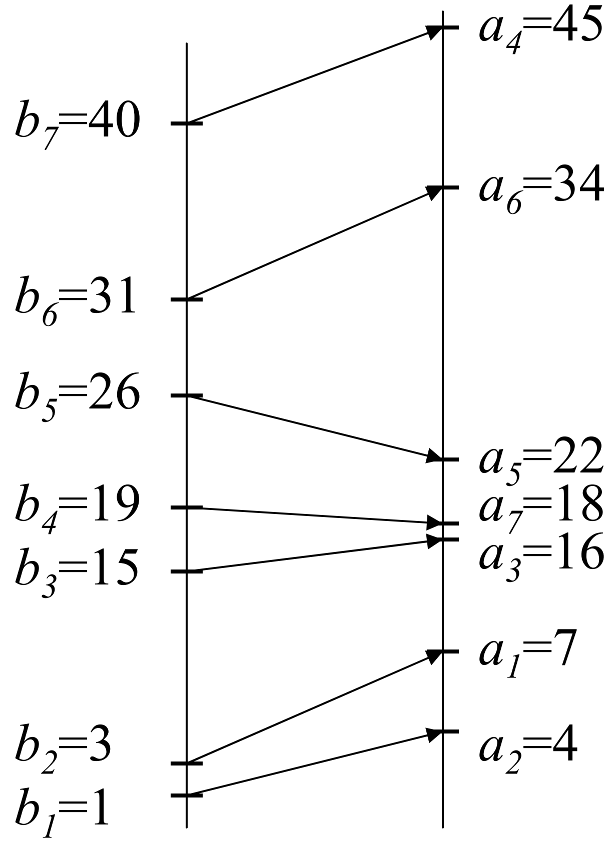
Arrange the b_j in order of size and renumber the jobs so that

$$b_1 \leq b_2 \leq \ldots \leq b_n$$

Arrange the a_j in order of size.

The permutation mapping ϕ^* is defined by

$$\phi^*(j) = k, \ k \text{ being such that } a_k \text{ is the } j\text{th smallest of the } a.$$

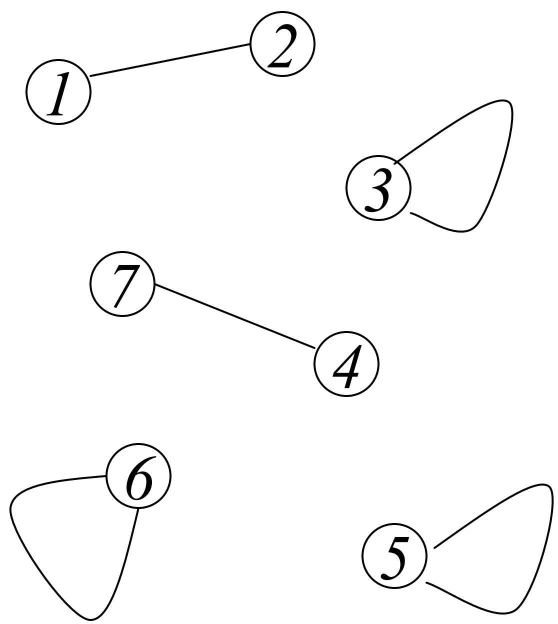


jobs	*1*	*2*	*3*	*4*	*5*	*6*	*7*
b_j	1	3	15	19	26	31	40
a_j	7	4	16	45	22	34	18
$a_{\phi^*}(j)$	4	7	16	18	22	34	45
$\phi^*(j)$	*2*	*1*	*3*	*7*	*5*	*6*	*4*

Step 2.

Form an undirected graph with n nodes and undirected arcs connecting the jth and $\phi^*(j)$ nodes, $j=1,\ldots n$.

If the current graph has only one component then STOP ; otherwise go to Step 3.



Step 3.

Compute the swap costs $C_{\phi^*} I(j, j+1)$ for j=1,…,n

$$C_{\phi^*} I(j, j+1) = 2 \max \big(\min (b_{j+1}, a_{\phi^*(j+1)}) - \max (b_j, a_{\phi^*(j)})), 0 \big)$$

$C_{\phi^*} I(1, 2) = 2 \max ((3\text{-}4), 0) = 0$

$C_{\phi^*} I(2, 3) = 2 \max ((15\text{-}7), 0) = 16$

$C_{\phi^*} I(3, 4) = 2 \max ((18\text{-}16), 0) = 4$

$C_{\phi^*} I(4, 5) = 2 \max ((22\text{-}19), 0) = 6$

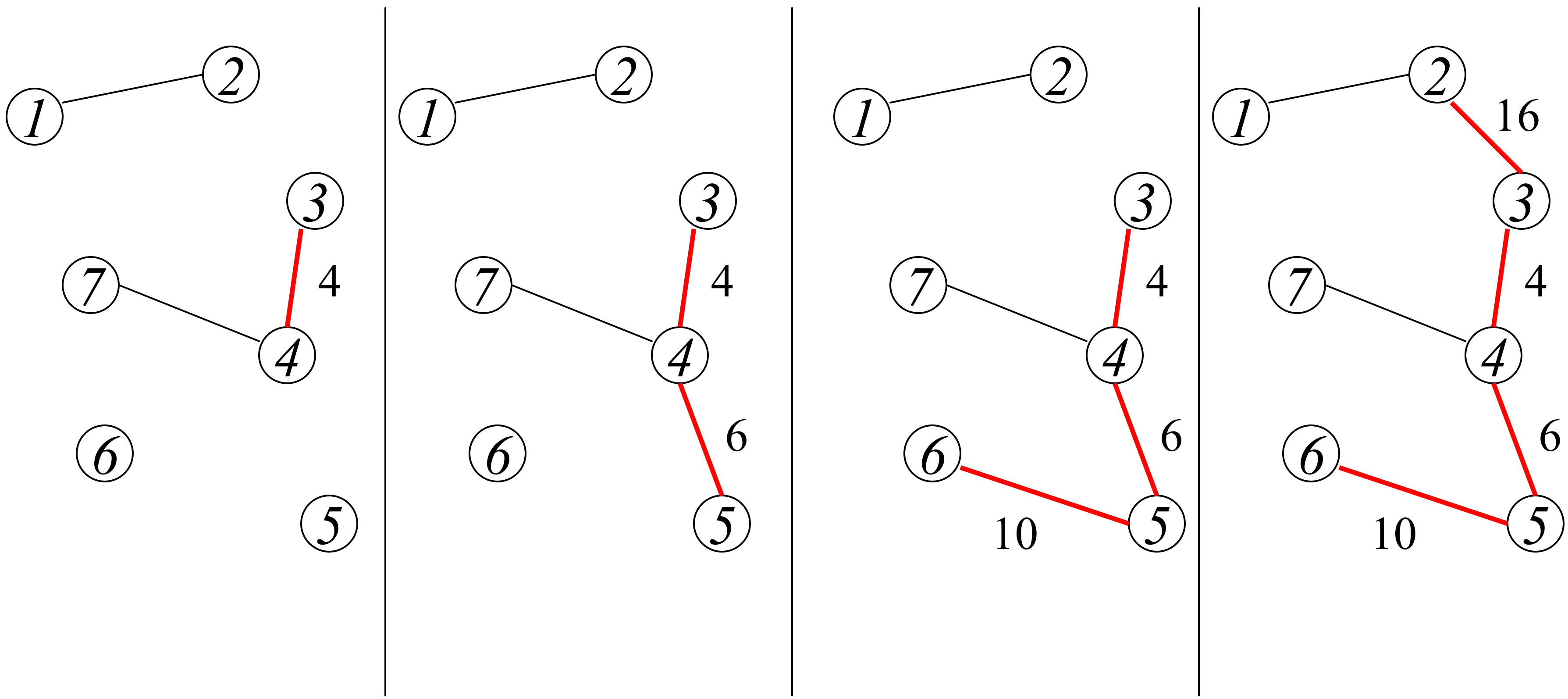
$C_{\phi^*} I(5, 6) = 2 \max ((31\text{-}26), 0) = 10$

$C_{\phi^*} I(6, 7) = 2 \max ((40\text{-}34), 0) = 12$

Step 4.

Select the smallest value $C_{\phi^*} I(j, j+1)$ such that j is in one component and $j+1$ in another. In case of a tie for smallest, choose any.

Insert the undirected arc $R_{j,\,j+1}$ into the graph. Repeat this step until all the components in the undirected graph are connected.



Step 5.

Divide the arcs added in Step 4 into two groups.
Those $R_{j,j+1}$ for which $b_j \le a_{\phi(j)}$ go in *group 1*,
those for which $b_j > a_{\phi(j)}$ go in *group 2*.

arcs	b_j	$a_{\phi^*(j)}$	*group*
$R_{2,3}$	b_2=3	a_1=7	1
$R_{3,4}$	b_3=15	a_3=16	1
$R_{4,5}$	b_4=19	a_7=18	2
$R_{5,6}$	b_5=26	a_5=22	2

Step 6.

Find the largest index j_1 such that $R_{j_1,\, j_1+1}$ is in *group 1*.
Find the second largest index, and so on, up to j_l assuming there are l elements in the group.

Find the smallest index k_1 such that $R_{k_1,\, k_1+1}$ is in *group 2*.
Find the second smallest index, and so on, up to k_m assuming there are m elements in the group.

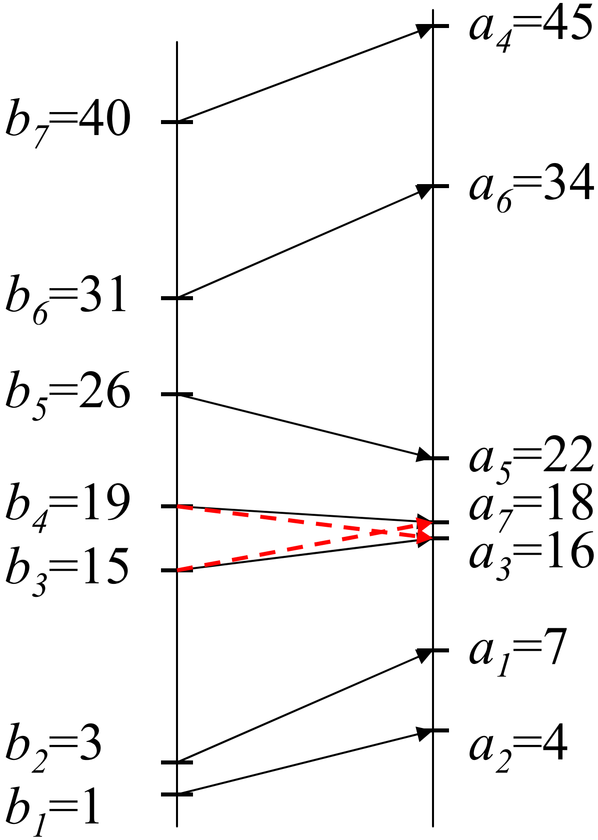
$j_1 = 3,\ j_2 = 2,$ $\qquad$ $k_1 = 4,\ k_2 = 5$

Step 7.

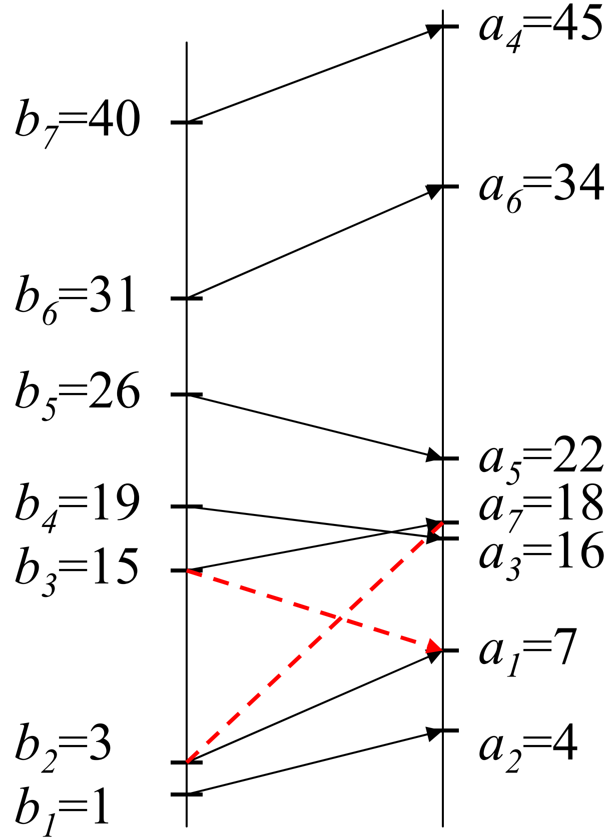
The optimal tour ϕ^{**} is constructed by applying the following sequence of swaps on the permutation ϕ^*:

$$\phi^{**} = \phi^* \, I(j_1, j_1+1)\, I(j_2, j_2+1) \ldots I(j_l, j_l+1)$$
$$I(k_1, k_1+1)\, I(k_2, k_2+1) \ldots I(k_m, k_m+1)$$

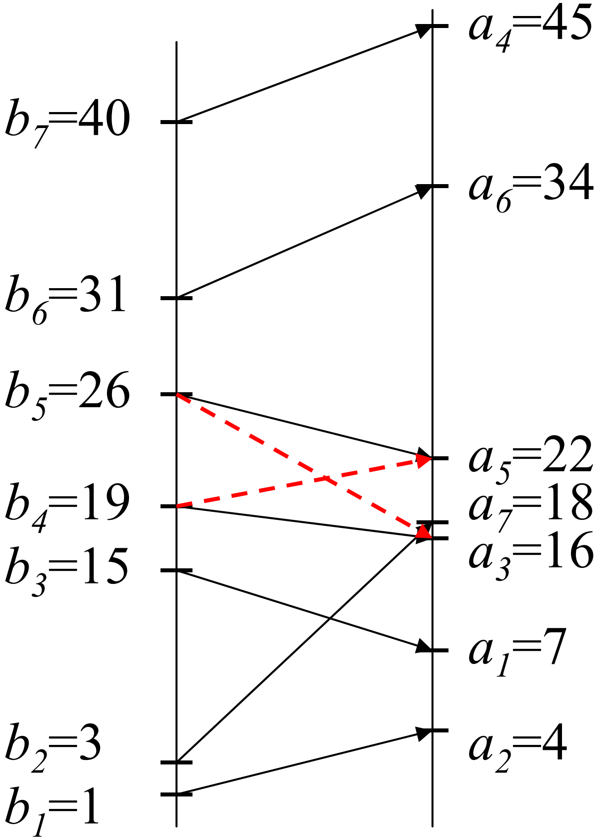
$$\phi^{**} = \phi^* \, \underbrace{I(3,4)\, I(2,3)}_{\text{Type 1}}\, \underbrace{I(4,5)\, I(5,6)}_{\text{Type 2}}$$



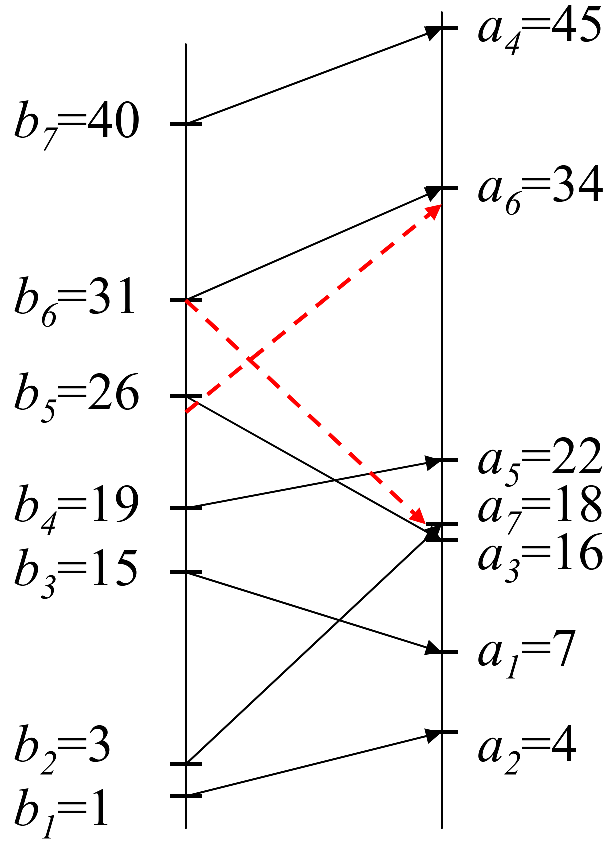
$\phi^*\ I(3,4)$



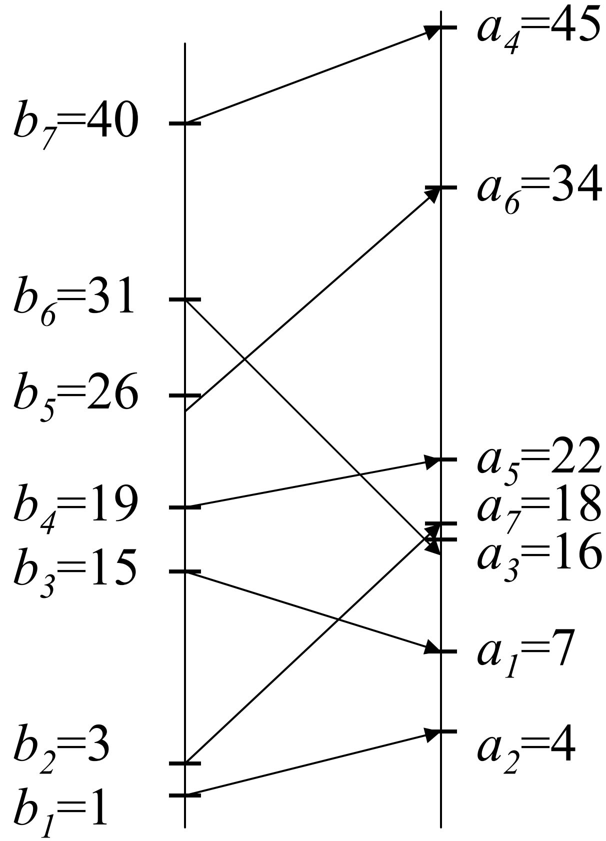
$\phi^*\ I(3,4)\ I(2,3)$



$\phi^* \, I(3,4) \, I(2,3) \, I(4,5)$



$\phi^{**} = \phi^* \, I(3,4) \, I(2,3) \, I(4,5) \, I(5,6)$



$$\phi^{**} = \phi^{*}\, I(3,4)\, I(2,3)\, I(4,5)\, I(5,6)$$

The <u>optimal tour</u> is: $1 \rightarrow 2 \rightarrow 7 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 3 \rightarrow 1$

The <u>cost</u> of the tour is: $3 + 15 + 5 + 3 + 8 + 15 + 8 = 57$