Chapter 3

Branch and Bound Algorithm for Single machine Earliness/Tardiness Problem with Sequence-Dependent Setup Cost

In this chapter, the single machine scheduling problem to minimize earliness/ tardiness penalties with sequence-dependent setup cost is presented. The network representation and branch and bound algorithm are proposed to determine a global optimal production schedule. The development of branch and bound algorithm for this problem is achieved in two steps. The early step is a simple branch and bound algorithm with heuristic methods to determine upper bound (B&B1). In the second step, an efficient lower and upper bounding procedure is incorporated into basic branch and bound algorithm (B&B2) in order to improve the computational efficiency.

3.1 Problem Description

Notations:

The following notations will be used in the following sections in this chapter.

X	set of N independent jobs, $X = \{x_1, x_2, ..., x_N\}$
N	total number of jobs
SC	sequence-dependent setup cost matrix, **SC** = $[SC_{ji}]$
SC_{ji}	setup cost of switching from job j to i
Rp_i	production rate
Q_i	demand of job i
d_i	due date of job i
p_i	processing time of job i
h_i	earliness weight of job i
w_i	tardiness weight of job i
E_i	earliness of job i, $E_i = max\ (0, d_i - C_i)$
T_i	tardiness of job i, $T_i = max\ (0, C_i - d_i)$.
y_{ji}	binary variable, $y_{ji} = 1$ if job j precedes job i; otherwise $y_{ji} = 0$.
λ	vector of corresponding lagrangian multipliers, $\lambda = (\lambda_1, \lambda_2, ..., \lambda_N)$
S_l	ordered set of scheduled jobs of each node at level l excluding the last job in the schedule. Note that at level 0, $S_0 = \varnothing$.
S_l'	set of unscheduled jobs of each node at level l.
$N - l$	number of elements in set S_l'
k	look-ahead parameter

X is a set of N independent jobs $\{x_1, x_2, ..., x_N\}$ which are to be scheduled non-preemptively on a machine that can handle at most one job at a time. Once the current job is finished, the next job is started immediately without an idle time. The sequence-dependent setup cost is described by a matrix **SC** = $[SC_{ji}]$, when SC_{ji} is the cost of

switching from job j to i. The setup cost of the first job, i.e. SC_{0i}, is assumed to be zero. The setup time is sequence-independent and is added to the processing time. Each job is available at time zero and its distinct due date d_i is known. The processing requirement or processing time p_i is expressed in days of production, which can be obtained from a demand of job i, Q_i, divided by its production rate Rp_i. The earliness and tardiness penalties per period are represented by h_i and w_i, respectively. If the completion time C_i is before the due date, the earliness of job i is determined from $E_i = max\ (0,\ d_i - C_i)$. If the job is completed after the due date, the tardiness of job i is then determined from $T_i = max\ (0,\ C_i - d_i)$.

3.2 Objective Function

The objective function of early/tardy problem with sequence-dependent setup cost can be written as (P):

$$\text{(P)} \qquad V = \text{Min} \sum_{i=1}^{N} \left(h_i E_i + w_i T_i + y_{ji} SC_{ji} \right) \qquad (3.1)$$

subject to

$$E_i \geq 0 \qquad (3.2)$$

$$E_i \geq d_i - C_i \qquad (3.3)$$

$$T_i \geq 0 \qquad (3.4)$$

$$T_i \geq C_i - d_i \qquad (3.5)$$

3.3 Branch and Bound Algorithm 1 (B&B1)

In the first step, Branch and Bound algorithm with heuristic methods is used to solve the problem. A node represents the sequence of jobs that are already scheduled. An arc represents the total cost of the node. The techniques of finding initial upper bound, branching and fathoming procedure are illustrated below.

3.3.1 Heuristic Procedure

One way to shorten the calculation is to find the initial feasible solution which will give an upper bound of the objective function. If the lower bound of a sub-problem is greater than or equal to the upper bound, then this sub-problem cannot yield a better solution, and hence we need not continue to branch from the corresponding node in the search tree. Two dispatching heuristics are used to find the initial solutions and their objective values. The minimum of the two objective values is the upper bound.

Heuristic 1: Minimum Setup

Step 1. Select the job with the earliest due date to process first. Tie breaking by selecting the job with less processing time.

Step 2. From the setup cost matrix, the next job to be scheduled is the job with the minimum setup cost from its preceding job. Tie breaking by selecting the job with earlier due date in order to avoid tardiness penalty.
Step 3. Continue step2 until all jobs all jobs are scheduled.
Step 4. Calculate the total cost of schedule and set it to be an initial minimum

Heuristic 2: Earliest Due Date

Step 1. Placing the job with earliest due date in the first position. Tie breaking by selecting the job with less processing time.
Step 2. The next job scheduled is the job that has the earliest due date from among the jobs that are yet to be scheduled. Tie breaking by selecting the job with less setup cost in order to minimize setup cost.
Step 3. Continue step2 until all jobs all jobs are scheduled.
Step 4. Calculate the total cost of schedule and set it to be an initial minimum value.

3.3.2 Step of Branch and Bound Algorithm 1

Branching and Fathoming Procedure

Step 1. Determine the job to be processed first. Draw a network diagram indicating a node for each possible job to be processed first. A node represents the sequence of jobs that are already scheduled.
Step 2. Calculate the total cost associated with each node and represented on the arc.
Step 3. Eliminate all nodes that have associated cost greater than or equal to the upper bound value obtained from the heuristic procedure. The remaining nodes are called active nodes.
Step 4. Select the active node with the lowest associated cost (parent node) for branching. Each of the remaining jobs can be processed in the next position. Construct nodes for each of these jobs and develop branches connecting these nodes from the parent node.
Step 5. Calculate the associated cost for each node developed in step 4 and added to the associated cost of the parent node to obtain the associated cost of the present node. Replace the current upper bound with objective value of the complete sequence if better value is founded.
Step 6. Eliminate nodes where associated cost is above the current upper bound. Repeat step 4 or stop calculation if there are no active node remained. The current node will be the optimal sequence.

3.3.3 Illustrative Example 1

The procedure of Branch and Bound algorithm 1 can be illustrated using the following numerical example. There are 4 jobs to be processed on a single machine, any one of which could be processed in any position in a sequence. The data are shown in Table 3.1. The setup costs are given by the elements c_{ji} in the matrix C, where

$$SC = \begin{pmatrix} 0 & 1 & 3 & 1 \\ 2 & 0 & 1 & 2 \\ 3 & 4 & 0 & 1 \\ 2 & 1 & 5 & 0 \end{pmatrix}$$

Table 3.1 Data for illustrative example 1

Jobs	Processing Time	Due Date	Earliness Penalty	Tardiness Penalty
1	4	7	2	4
2	3	2	1	2
3	1	5	3	3
4	2	10	2	8

From the minimum setup heuristic, the sequence of 2, 3, 4, 1 is obtained with the total cost of 29. The EDD rule gives the sequence of 2, 3, 1, 4 with the total cost of 23. Therefore, the upper bound for the problem is initially set to be 23.

In the first iteration 1, nodes 1, 2, 3, and 4 are constructed, indicating that each of the four jobs could be processed in the first position in the sequence. The total penalty costs are calculated and are shown on the arcs leading to those nodes. Select job 2 as the branching node because job 2 has the lowest penalty value of the four nodes. Job 2 is now assigned to position 1 in the sequence. Therefore, branching from job 2 for job 1, 3, and 4, placing each one of job in the second position. Then the total cost for each node is calculated. Since node (2,1) has the lowest associated cost, placing job 1 in the second position. Branching from node (2,1) for the remaining jobs, which are job 3 and 4. Since node (2,1,3) has the lower cost, branching from this node to obtain the initial sequence of (2,1,3,4) with the total penalty cost of 17. The initial total penalty cost obtained from the completed sequence is less than the upper bound, thus set this value to be the current upper bound. Search all existing nodes available in order to bound, and fathom any node with total penalty value higher than the upper bound. The last unfathomed node will be the optimal sequence

3.4 Branch and Bound Algorithm 2 (B&B2)

In the second step, Branch and Bound algorithm with efficient lower and upper bounding procedure is developed. Lower bounds are obtained based on heuristic and Lagrangian relaxation. Priority dispatching rule with local improvement procedure is used to derive an initial upper bound. Two dominance criteria are incorporated in a Branch and Bound procedure to reduce the search space and enhance computational efficiency.

3.4.1 Lower Bound Derivation

In this section, procedures for deriving lower bounds are presented. The original problem (P) is decomposed into three sub-problems: weighted early (P1), weighted tardy (P2), and sequence-dependent setup (P3).

Sub-problem P1, weighted earliness sub-problem

$$\text{(P1)} \qquad V_1 = \text{Min} \sum_{i=1}^{N} h_i E_i \tag{3.6}$$

subject to

$$E_i \geq 0 \tag{3.7}$$
$$E_i \geq d_i - C_i \tag{3.8}$$

Sub-problem P2, weighted tardiness sub-problem

$$\text{(P2)} \qquad V_2 = \text{Min} \sum_{i=1}^{N} w_i T \tag{3.9}$$

subject to

$$T_i \geq 0 \tag{3.10}$$
$$T_i \geq C_i - d_i \tag{3.11}$$

Sub-problem P3, sequence-dependent setup sub-problem

$$\text{(P3)} \qquad V_3 = \text{Min} \sum_{i=1}^{N} y_{ji} SC_{ji} \tag{3.12}$$

The lower bounds of P1, P2, and P3 can be used to obtain the lower bound for P. The lower bounds of P1 and P2 are computed based on Lagrangian relaxation, and a multiplier adjustment method is used to compute the value of Lagrange multipliers. The lower bound of P3 is obtained using a minimum setup heuristic. The lower bounding procedures developed by Li (1997) are used to obtain the lower bounds for P1 and P2. The heuristic used to obtain the lower bound for P3 is developed in this paper. The lower bound for the original problem is the sum of the three lower bounds.

Lower bound for sub-problem P1

A Lagrangian relaxation of constraint (3.8) yields the Lagrangian problem LR1:

$$\text{(LR1)} \qquad L_1(\lambda) = \text{Min} \sum_{i=1}^{N} \left[(h_i - \lambda_i) E_i + \lambda_i (d_i - C_i) \right] \tag{3.13}$$

subject to

$$E_i \geq 0 \tag{3.14}$$

In (LR1), the objective function can be expanded as follows:

$$L_1(\lambda) = \text{Min} \sum_{i=1}^{N} (h_i - \lambda_i) E_i + \sum_{i=1}^{N} \lambda_i d_i - \text{Max} \sum_{i=1}^{N} \lambda_i C_i \tag{3.15}$$

According to Li (1997), for any choice of nonnegative multipliers, $L_1(\lambda)$ provides a lower bound for problem P1. The first term can be minimized by setting all $E_i = 0$ if $h_i - \lambda_i \geq 0$. The second term is a constant. The third term can be maximized using the weighted longest processing time rule (WLPT), which is sequencing jobs in non-decreasing order of λ_i/p_i. Assume that the jobs generated by the heuristic presented below are renumbered so that the sequence is (1, 2,…, N). Also, let C_i* be the completion time of jobs when they are sequenced by the heuristic. Then, a lower bound $L_1(\lambda)$ can be determined from the Lagrangian dual sub-problem D1.

(D1)
$$L_1 = \text{Min} \sum_{i=1}^{N} \lambda_i \left(d_i - C_i^* \right) \tag{3.16}$$

subject to

$$\frac{\lambda_i}{p_i} \leq \frac{\lambda_{i+1}}{p_{i+1}} \qquad i = 1,....., N-1 \tag{3.17}$$

$$0 \leq \lambda_i \leq h_i \qquad i = 1,....., N-1 \tag{3.18}$$

The multiplier adjustment method developed by Potts and Wassenhove (1985) requires the following heuristic to sequence the jobs. Then, Li (1997) used this method to solve the problem. The multipliers obtained from the heuristic can solve the Lagrangian problem so that the resulting lower bound is as large as possible.

Heuristic for solving Lagrangian dual sub-problem D1

Step1. Sequence jobs by WLPT rule. Set $E_i = d_i - C_i^*$, $i = 1,\ldots, N$ and
$V_i = \sum_{k=i}^{N} p_k E_k$, $i = 1,..., N$

Step 2. Set $V_{N+1} = 0$, $S_1 = \{N + 1\}$, and $k = N$
Step 3. For $k = N$ to 1, when $k < 1$ go to step 4
Let m be the smallest integer in set S_1
If $V_k > V_m$, then include k in set S_1, i.e., $S_1 = S_1 \cup \{k\}$
Set $k = k - 1$
Step 4. Set $k = 1$, and $S_1 = S_1 - \{ N + 1\}$
Step 5. For $k = 1$ to N, when $k > N$ terminate
If $k = 1$ and $k \notin S_1$, then $\lambda_k = 0$
If $k > 1$ and $k \notin S_1$, then $\lambda_k = \lambda_{k-1}$ (p_k/p_{k-1})
If $k \in S_1$, then $\lambda_k = h_k$
$k = k + 1$

The Lagrangian multiplier (λ_i) and C_i* can be obtained from the above heuristic. Then L_1 can be calculated and used as a lower bound $LB1$ for sub-problem P1.

Lower bound for sub-problem P2

A Lagrangian relaxation of constraint (3.11) yields the Lagrangian problem LR2:

(LR2) $$L_2(\lambda) = \text{Min} \sum_{i=1}^{N} [(w_i - \lambda_i)T_i + \lambda_i(C_i - d_i)] \quad (3.19)$$

subject to

$$T_i \geq 0 \quad (3.20)$$

Similar to LR1, LR2 can be solved using the weighted shortest processing time rule (WSPT) by setting $T_i = 0$ if $w_i - \lambda_i \geq 0$. Then, the lower bound $L_2(\lambda)$ can be obtained from D2.

(D2) $$L_2 = \text{Min} \sum_{i=1}^{N} \lambda_i (C_i^* - d_i) \quad (3.21)$$

subject to

$$\frac{\lambda_i}{p_i} \geq \frac{\lambda_{i+1}}{p_{i+1}} \qquad i = 1,....., N-1 \quad (3.22)$$

$$w_i \geq \lambda_i \geq 0 \qquad i = 1,....., N-1 \quad (3.23)$$

Similarly, the multiplier adjustment method is used to solve the problem.

Heuristic for solving Lagrangian dual sub-problem D2

Step1. Sequence jobs by the WSPT rule. Set $T_i = C_i^* - d_i$, $i = 1,\ldots,N$ and

$V_i = \sum_{k=1}^{i} p_k T_k, i = 1,...,N$

Step 2. Set $V_0 = 0$, $S_2 = \{0\}$, and $k = 1$

Step 3. For $k = 1$ to N, when $k > N$ go to step 4
Let m be the largest integer in set S_2
If $V_k > V_m$, then include k in set S_2, i.e., $S_2 = S_2 \cup \{k\}$
Set $k = k + 1$

Step 4. Set $k = N$, and $S_2 = S_2 - \{0\}$

Step 5. For $k = N$ to 1, when $k < 1$ terminate
If $k = N$ and $k \notin S_2$, then $\lambda_k = 0$
If $k < N$ and $k \notin S_2$, then $\lambda_k = \lambda_{k+1} (p_k/p_{k+1})$
If $k \in S_2$, then $\lambda_k = w_k$
$k = k - 1$

The Lagrangian multiplier (λ_i) and C_i* can be obtained from the above heuristic. Then L_2 can be calculated and used as a lower bound $LB2$ for sub-problem P2.

Lower bound for sub-problem P3

A Lower bound of sub-problem P3 can be obtained by using minimum setup heuristic developed in this research.

Minimum setup heuristic

Step 1. For each node, determine set S_l and S'_l.

Step 2. For $i = 1$ to $N - l$, Calculate the minimum sequence-dependent setup cost (MSC_{x_i}) of each element x_i.

$$MSC_{x_i} = \min_{\substack{x_j \neq x_i \\ x_j \notin S_l \\ x_i \in S'_l}} (SC_{x_j x_i}) \tag{3.24}$$

where, $SC_{x_j x_i}$ is the sequence-dependent setup cost from job x_j to x_i.

Step 3. Calculate the minimum sequence-dependent setup cost of the node (MSC_l)

$$MSC_l = \sum_{i=1}^{N-l} MSC_{x_i} \tag{3.25}$$

Then, the MSC_l is used as a lower bound LB3 for sub-problem P3.

3.4.2 Upper Bound Derivation

In this section, two-phase heuristic procedures are used to determine an initial feasible solution for the problem. The priority dispatching rule developed by Ow and Morton (1989) is used in the first phase to establish an initial sequence. In the second phase, the local improvement procedure developed by Liaw (1999) is modified to take into account the sequence-dependent setup cost. The modified procedure is used to adapt the sequence in the first phase to achieve a better schedule and a tighter upper bound. The improvement is obtained by an insertion procedure followed by a swap procedure.

Priority dispatching rule

Step 1. Calculate the slack of each job i at time t, $s_i = d_i - t - p_i$.

Step 2. Calculate average processing time ($\bar{p}$) of all remaining jobs.

Step 3. Calculate priority index $I_i(t)$ at time t of each remaining job using Equation 3.26.

Step 4. Choose the job with the highest index to process next in the sequence.

Step 5. Repeat steps 1 to 4 until all jobs are included in the schedule.

Local improvement procedure

Insertion procedure

Step 1. At iteration j, $j = 1, 2, \ldots, N$, select job A as the job in position j.

$$I_i(t) = \begin{cases} \dfrac{w_i}{p_i} & \text{if} \quad s_i \le 0 \\ \dfrac{w_i}{p_i}\exp\left[-\dfrac{(h_i+w_i)s_i}{h_i\overline{p}}\right] & \text{if} \quad 0 < s_i \le \left(\dfrac{w_i}{w_i+h_i}\right)k\overline{p} \\ h_i^{-2}\left[\dfrac{w_i}{p_i} - \dfrac{(h_i+w_i)s_i}{k\,p_i\,\overline{p}}\right]^3 & \text{if} \quad \left(\dfrac{w_i}{h_i+w_i}\right)k\overline{p} < s_i < k\overline{p} \\ -\dfrac{h_i}{p_i} & \text{otherwise} \end{cases} \tag{3.26}$$

Step 2. Select job B among [*N*/3] jobs nearest to job A. The candidate for job B is restricted since it seldom occurs that the best job B candidate is far away from job A selected.
Step 3. Calculate the objective value (total cost of early/ tardy penalties and setup cost) after inserting job A before job B and compare with each one of the [*N*/3] cases.
Step 4. Select job B such that the resulting schedule has the minimum objective value.
Step 5. Insert job A before job B.
Step 6. Repeat steps 1 to 5 until $j = N$

Swap procedure

Steps 1-2. They are the same as those of the insertion procedure.

Step 3. Compute the objective value after interchanging job A and job B and compare with each one of the [*N*/3] cases.
Step 4. Select job B such that the resulting schedule has the minimum objective value.
Step 5. Swap jobs A and B
Step 6. Repeat steps 1 to 5 until $j = N$.

3.4.3 Dominance Criteria

In this section, the dominance criteria specially formulated for early/tardy problem with sequence-dependent setup cost are presented. The criteria are an extension of precedence relations developed by Ow and Morton (1989) and Liaw (1999) so that they can incorporate sequence-dependent setup cost. Dominance criteria consist of adjacency condition and non-adjacency condition, in which any adjacent and non-adjacent job must satisfy these conditions, respectively. Any node which does not satisfy these conditions will be removed from a branch and bound search tree.

Adjacency Condition

Suppose that jobs *i* and *j* are adjacent pairs of jobs before the last position and jobs *i* and *j* lie between jobs *x* and *y* as shown in Figure 3.1. The total sequence-dependent setup cost, TSC_{ij} and TSC_{ji}, can be calculated using formulas (3.27) and (3.28)

.

$$TSC_{ij} = SC_{xi} + SC_{ij} + SC_{jy} \tag{3.27}$$

$$TSC_{ji} = SC_{xj} + SC_{ji} + SC_{iy} \tag{3.28}$$

when job *i* immediately precedes job *j*, Ω_{ij} and Ω_{ji} are defined as:

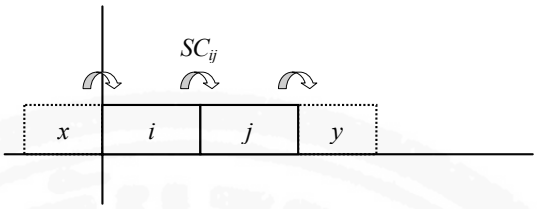

Figure 3.1 Illustration of adjacency condition of jobs i and j.

$$\Omega_{xy} = \begin{cases} 0 & if \quad s_x \le 0 \\ s_x & if \quad 0 < s_x < p_y \\ p_y & if \quad s_x \ge p_y \end{cases} \tag{3.29}$$

where, $s_x = d_x - t - p_x$ is the slack of job x.
t = the sum of processing times of all preceding jobs.

The following condition must hold for all adjacent pairs of jobs:

$$AJC_{ij} = w_i p_j - \Omega_{ij}(w_i + h_i) + TSC_{ji} \tag{3.30}$$

$$AJC_{ji} = w_j p_i - \Omega_{ji}(w_j + h_j) + TSC_{ij} \tag{3.31}$$

$$AJC_{ij} \ge AJC_{ji} \tag{3.32}$$

See the proof of adjacency condition in Appendix A.

There are many nodes that are generated from the same parent node. Thus, calculating the adjacency condition using formulas (3.27) to (3.32) for all nodes is time consuming. It is possible to reduce the computation time by calculating the adjacency condition only for the first node and using formula (3.33) for other nodes generated from the same parent node.

$$AJC_{ij} + \Delta SC_{jy} \ge AJC_{ji} + \Delta SC_{iy} \tag{3.33}$$

where, ΔSC_{jy} = SC_{jy} of the node under consideration - SC_{jy} of the first node
ΔSC_{iy} = SC_{iy} of the node under consideration - SC_{iy} of the first node.

Proof: Consider any node generated from the same parent node. Jobs *i* and *j* are the same; hence, $w_i p_j - \Omega_{ij}(w_i + h_i)$ is equal to $w_j p_i - \Omega_{ji}(w_j + h_j)$. The value of TSC_{ij} and TSC_{ji} are varied according to the last job. The only differences are SC_{jy} and SC_{iy}. Consequently, the adjacency condition can be computed by adding only the different values (ΔSC_{jy} and ΔSC_{iy}) to the same formula.

The adjacency condition can be applied to the node at level 3 and higher. The reason can be shown by the following example. Suppose the node in level 2 is considered, e.g., node (1, 2). Since the setup cost is sequence-dependent, from formulas (3.27) and

(3.28), SC_{jy} and SC_{iy}, cannot be determined. But if the node is branched further to level 3, e.g. node (1, 2, 3), in this case SC_{jy} is SC_{23} and SC_{iy} is SC_{13}.

Non-Adjacency Condition

Suppose jobs i and j are non-adjacent pairs of jobs before the last position and they have the same processing time. Job i lies between jobs u and v and job j lies between jobs w and z as shown in Figure 3.2. The total sequence-dependent setup cost, TSC_{ij} and TSC_{ji}, can be calculated using formulas (3.34) and (3.35).

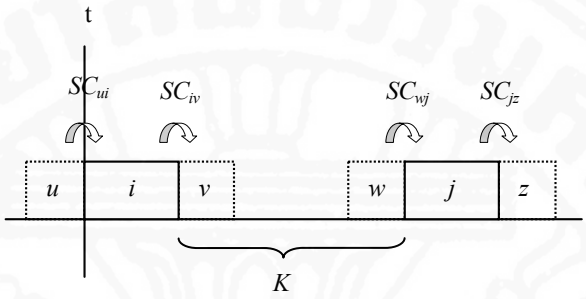

Figure 3.2 Illustration of non-adjacency condition of jobs i and j.

$$TSC_{ij} = SC_{ui} + SC_{iv} + SC_{wj} + SC_{jz} \quad (3.34)$$

$$TSC_{ji} = SC_{uj} + SC_{jv} + SC_{wi} + SC_{iz} \quad (3.35)$$

Δ_{ij} and Δ_{ji} are defined as:

$$\Delta_{xy} = \begin{cases} 0 & if \quad s_x \le 0 \\ s_x & if \quad 0 < s_x < p_y + K \\ p_y + K & if \quad s_x \ge p_y + K \end{cases} \quad (3.36)$$

where, $s_x = d_x - t - p_x$ is the slack of job x
K = the sum of processing times of jobs between jobs i and j

All non-adjacent pairs of jobs in the optimal schedule must satisfy the following condition:

if $p_i = p_j$, then

$$NJC_{ij} = w_i (p_j + K) - \Delta_{ij} (w_i + h_i) + TSC_{ji} \quad (3.37)$$

$$NJC_{ji} = w_j (p_i + K) - \Delta_{ji} (w_j + h_j) + TSC_{ij} \quad (3.38)$$

$$NJC_{ij} \ge NJC_{ji} \quad (3.39)$$

See the proof of non-adjacency condition in Appendix B.

The non-adjacency condition can be applied to the node at level 4 and higher. In other words, at least four jobs must be in the sequence. For example, non-adjacent jobs of

the node at level 4 are jobs in positions 1 and 3. The last job in position 4 is used to determine the sequence-dependent setup cost.

3.4.4 Step of Branch and Bound Algorithm 2

A network representation and branch and bound algorithm can be used to determine an optimal solution for the problem. A node represents a sub-problem or the sequence of jobs that are already scheduled. An arc leading to the node represents the associated cost of the node.

Initialization step

The branch and bound algorithm is started by determining an initial feasible solution to the problem. According to the upper bound (or two-phase heuristic) procedures presented in section 3.4.2, the priority-dispatching rule is applied to establish the initial sequence. The local improvement procedures are used to modify the schedule obtained in the first phase in order to get a better sequence. The objective function value obtained from the two-phase heuristic is an initial upper bound of the objective function value.

Steps for each iteration

Step 1. Branching

The branching strategy is called the depth-first branching rule. Select the active (unfathomed) node at the highest level in the search tree for branching. If there is more than one node at the highest level, select the one with the lowest associated cost. The selected node is called the parent node. Create branches from the parent node to the nodes representing jobs that have not been scheduled.

Step 2. Bounding

For each newly created node, the associated cost is calculated. The associated cost is the sum of the associated cost of its parent node and the cost due to the newly assigned job of that node.

Step 3. Fathoming

For each generated node, three computational tests are performed in the fathoming step. First, a node is fathomed if its associated cost is greater than or equal to the current upper bound of the problem (denoted by F_1). Second, the dominance criteria described in section 3.4.3 are applied to further reduce the number of nodes (denoted by F_2). Dominance criteria consist of adjacency and non-adjacency conditions. All pairs of jobs in the optimal schedule must satisfy these conditions. Note that the adjacency condition can be applied to the nodes at level 3 and higher, while the non-adjacency condition can be applied to the nodes at level 4 and higher. If the node is not eliminated by the above two tests, the lower bounds of each of three sub-problems are computed according to the procedures presented in section 3.4.1. For fast elimination of nodes, the lower bounds of three sub-problems can be calculated in any sequence according to the characteristics of the problem under consideration. Suppose that the sequence of lower bound is $LB1$, $LB2$, and MSC. When the $LB1$ is determined, it is summed to the associated cost of the node and compared with the upper bound. If it is greater than or equal to the current upper bound, the node is fathomed (denoted by F_3). If the node is still not fathomed, the $LB2$

will be computed. The value of $LB2 + LB1$ + associated cost of the node will be again compared to the upper bound. If the node is still unfathomed, the *MSC* will be determined. Finally, the sum of lower bounds of three sub-problems ($MSC + LB2 + LB1$) plus the associated cost of that node is compared to the upper bound.

The remaining unfathomed nodes are called active nodes. When a complete sequence is found (all jobs are scheduled on the node), the node is fathomed (denoted by F_4), and its objective function value will be compared with the current upper bound. If the current upper bound is higher, replace the current upper bound by the objective function value of the complete sequence.

Optimality test

If some active nodes are still available, repeat the iteration steps; otherwise, stop. After the algorithm is terminated, the node in which its objective function value is equal to the current upper bound represents the optimal solution and its objective function value is the optimal total costs.

3.4.5 Illustrative Example 2

The branch and bound algorithm can be illustrated using the following example. There are 4 jobs to be processed on a single machine. The data are shown in Table 3.2.

$$\boldsymbol{SC} = \begin{pmatrix} 0 & 15 & 33 & 3 \\ 20 & 0 & 18 & 28 \\ 35 & 8 & 0 & 10 \\ 2 & 14 & 5 & 0 \end{pmatrix}$$

Table 3.2 Data for illustration example 2

Jobs	Processing Time	Due Date	Earliness Penalty	Tardiness Penalty
1	3	8	2	9
2	2	6	7	7
3	4	10	3	8
4	3	9	8	6

The priority-dispatching rule generates the sequence of {1, 2, 4, 3} with the total cost of 89. The local improvement procedure modifies the sequence to be {2, 1, 4, 3} with a better total cost of 86, which is used as an initial upper bound for the problem.

Figure 3.3 shows a complete network representation of the illustrative example. In iteration 1, nodes 1, 2, 3, and 4 are constructed, indicating that each of the four jobs could be processed in the first order of the sequence. The associated cost of each node is calculated and shown on an arc leading to that node. Then, the associated cost is compared to the upper bound. The node will be fathomed if the associated cost is higher than or equal to the upper bound.

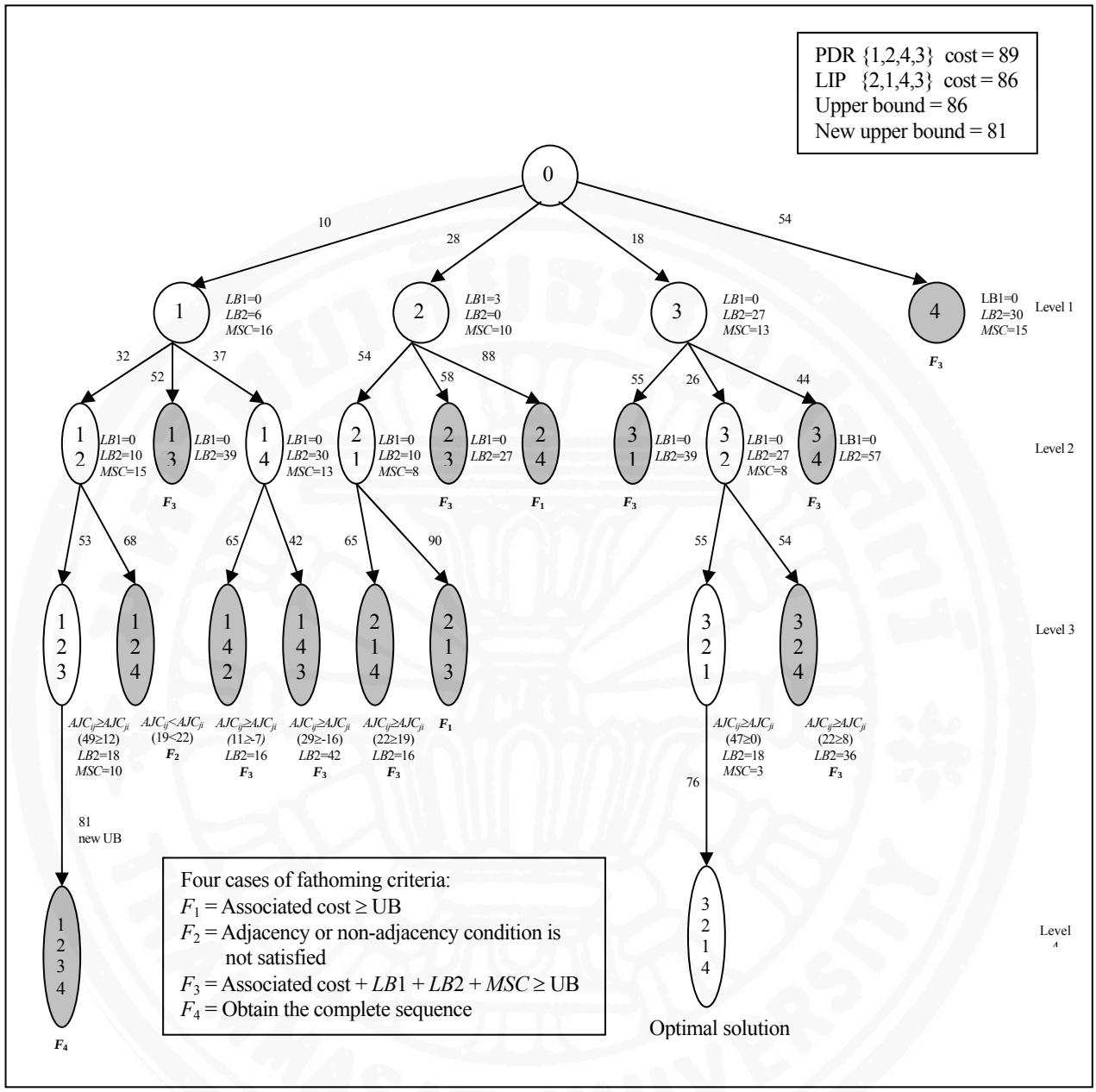

Figure 3.3 Network representation of illustrative example 2

In order to further eliminate these nodes, the lower bounds of the three sub-problems are computed. Each time the lower bound is obtained, it is summed up with the associated cost and compared to the upper bound. If it is greater than or equal to upper bound, the node is fathomed. In this case, node {4} is fathomed.

Then, select job 1 as the branching node because job 1 has the lowest objective function value among the three nodes. Job 1 is now assigned to the first order of the sequence. Then, branch from job 1 for jobs 2, 3, and 4, and place each of jobs 2, 3, and 4 in the second order of the sequence. The associated cost and lower bounds of each node are calculated. Node {1,3} has the associated cost plus lower bounds ($LB1$+$LB2$) greater than the upper bound. Thus it is fathomed (making it an inactive node). Since node {1,2} has the lowest cost, place job 2 in the second order. Branch from node {1,2} for the remaining jobs, which are jobs 3 and 4. At level 3, the dominance rule is applied. Since

node {1,2,4} does not satisfy the adjacency condition, the node is fathomed. Branch from the node {1,2,3} to obtain the first complete sequence of {1,2,3,4} with the objective function value of 81. The obtained objective function value is less than the current upper bound; thus, set the new upper bound equal to the objective function value. Fathom all nodes that have their associated costs greater than the new upper bound. Repeat the iteration steps until all nodes are fathomed. After the algorithm is terminated, the optimal sequence is {3,2,1,4} with the objective function value of 76.

3.5 Analysis of Computational Performance

The proposed branch and bound algorithms, B&B1 and B&B2, are implemented and computationally tested on a Pentium IV computer, 1.8 GHz CPU speed, 512MB RAM. Both algorithms are coded in C language. The test problems and analysis are divided into two parts, which are test of B&B1 and test of B&B2.

3.5.1 Experiment 1: Test of B&B1

For the test of B&B1, seven test problems with different number of jobs (4, 6, 8, 10, 12, 14, and 16 jobs) are randomly generated based on the parameters as shown in Table 3.3.

Each problem is tested by 30 randomly generated data sets. The performance measures are computational time and efficiency. The computational efficiency represents the efficiency of the fathoming step of the branch and bound algorithm. If most generated nodes can be fathomed very early, the algorithm will be terminated (the global optimal solution will be obtained) very quickly. The computational efficiency can be measured by *EFF*1 and *EFF*2. Tables 3.4 and 3.5 present the computational time, and the *EFF*1 and *EFF*2, respectively.

Table 3.3 Parameter setting for test of B&B1

Parameters	Range of value
ST_{ij}	*C* [0, 1]
SC_{ij}	*Int* [1, 5]
p_i	*Int* [1, 7]
d_i	*Int* [1, 4.5*N*]
h_i	*Int* [2, 5]
w_i	*Int* [5, 10]

C [a, b] = a set of real numbers on the interval [a, b] and
Int [a, b] = a set of integer numbers on the interval [a, b]

Table 3.4 shows that the required computational time is increasing exponentially with the problem size (number of jobs). This is an indication of an NP hard problem. The proposed algorithm can solve the problem up to 16 jobs in a reasonable time (3.4 hours on average) using PC. If the problem size is greater than 16 jobs, the required computational time will be unacceptable.

Table 3.5 shows that the *EFF*1 and *EFF*2 are increasing with the number of jobs. When the number of jobs is at least 10, the *EFF*1 and *EFF*2 are higher than 93% and 99%, respectively. This indicates that the branch and bound algorithm is very efficient in

fathoming the nodes. Although, the algorithm is very efficient, it cannot solve the problem having more than 16 jobs in reasonable time because of the *NP*-hard nature of the problem.

Table 3.4 Performance measures by computational time of B&B1

No. of jobs	Computational time (sec)		
	Average	Min	Max
4	0	0	0
6	0	0	0
8	0.73	0	1
10	11.13	0	52
12	64.62	0	561
14	2,130.54	1,072	4,342
16	15,430.16	10,465	21,560

Table 3.5 Computational efficiencies of B&B1

No. of jobs	Avg. no. of generated nodes	Avg. value of *EFF*1	Std. Dev. of *EFF*1	Avg. no. of generated completed solutions	Average value of *EFF*2	Std. Dev. of *EFF*2
4	35.27	44.93	17.78	7.13	70.278	19.40
6	$5.941*10^3$	66.28	18.89	69.06	90.38	7.19
8	$3.096*10^4$	71.75	21.93	2,466.86	93.88	7.49
10	$6.257*10^5$	93.52	10.74	14,206.06	99.60	0.71
12	$1.362*10^7$	99.66	0.67	30,755.27	99.99	0.157
14	$2.32*10^{11}$	98.12	0.48	128,179.53	99.99	0.13
16	$2.08*10^{13}$	99.47	0.35	827,745.36	99.99	0.098

EFF1 is the percentage of number of visited nodes by total number of generated nodes
EFF2 is the percentage of number of visited solutions by total number of feasible solutions.

3.5.2 Experiment 2: Test of B&B2

The proposed branch and bound algorithm with efficient upper and lower bounding procedure is implemented and computationally. The algorithms are coded in C language. The test problems are randomly generated based on the parameters as shown in Table 3.6. For each job, the values of p_i, h_i, and w_i, are the same as those in Li (1997) and Liaw (1999). The setup costs are carefully selected to ensure that they are not dominated by early/tardy penalties. For due date setting, Liaw (1999) analyzed the influence of average lateness factor and relative range of due date on problem hardness. The results indicated that the problems are most difficult when a lateness factor (*LF*) = 0.6, and increasing the range of due date (*RDD*) seems to make the problem harder. For the test problems, *LF* and *RDD* are equal to 0.5 and 1, respectively, which result in high problem hardness. The look-ahead parameter (k) is equal to 3 according to a recommendation of Liaw (1999).

Table 3.6 Parameter settings for the test problems.

Parameters	Range of value
SC_{ji}	*Int U*[1, 40]
p_i	*Int U*[1, 10]
d_i	*Int U*[$P(1- LF - RDD/2)$, $P(1 - LF + RDD/2)$] where, $LF = 0.5$ and $RDD = 1$
h_i	*Int U*[1, 10]
w_i	*Int U*[1, 10]
k	3

Int U[a, b] = a set of integer numbers generated from a discrete uniform distribution within an interval [a, b]
LF = Lateness factor
RDD = Range of due date

The first test (Test 1) evaluates the performance of the algorithm against the number of jobs. Six test problems with different numbers of jobs (15, 20, 25, 30, 35, and 40 jobs) are randomly generated. Each problem is tested by 30 randomly generated data sets. To evaluate the performance of upper bound, lower bounds, and dominance criteria, the second test (Test 2) compares the performance of the algorithm with and without implementation of these bounds and criteria for the problem of 20 and 30 jobs with the same parameter settings. For both tests, the program will terminate when the problem is not solved within 3,600 seconds in order to avoid excessive computational time. The number of unsolved problems is reported.

Table 3.7 Computational time required by the test problems (Test1)

No. of jobs	Avg. CPU time (sec)	Avg. no. of generated nodes	Avg. no. of generated completed solutions	Total no. of nodes	Total no. of solutions	No. of problems unsolved (out of 30)
15	0.413	13,838.34	67.83	$3.55 * 10^{12}$	$1.307 * 10^{12}$	0
20	4.48	94,351.67	161.27	$6.61 * 10^{18}$	$2.43 * 10^{18}$	0
25	37.67	$1.17 * 10^{6}$	925.34	$4.21* 10^{25}$	$1.55 * 10^{25}$	0
30	98.66	$6.96 * 10^{6}$	2,827.54	$7.21 * 10^{32}$	$2.65 * 10^{32}$	0
35	885.26	$4.87 * 10^{7}$	17,410.67	$2.808 * 10^{40}$	$1.03 * 10^{40}$	4
40	2847.33	$2.83 * 10^{8}$	89,455.37	$2.21 * 10^{48}$	$8.15 * 10^{47}$	10

The performance measures are computational time, number of generated nodes, and number of generated complete solutions. Table 3.7 and Figure 3.4 show that the required computational time increases exponentially with the problem size (number of jobs). This is an indication of an NP-hard problem. The propose algorithm can solve the problem having up to 40 jobs in a reasonable time (2,847 seconds on the average, not including unsolved problems). Comparing between the average number of generated nodes and the total number of nodes, and between the average number of generated complete solutions and the total number of solutions, it is obvious that the algorithm is very efficient in fathoming the nodes.

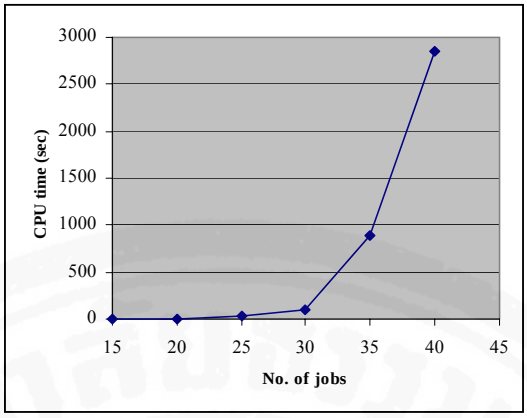

Figure 3.4 Relationship between the computational time and number of jobs.

Table 3.8 Comparison of efficiencies of each algorithm feature (Test 2)

Algorithm settings	No. of jobs	Avg. CPU time (sec)	Avg. no. of generated nodes	Avg. no. of generated completed solutions	No. of problems unsolved (out of 30)
UB is not used	20	7.75	391,702.5	899.75	0
	30	349.17	$1.34 * 10^7$	126,399.57	1
Dominance criteria are not used	20	28.12	963,951.42	195.73	0
	30	1426.18	$8.27 * 10^7$	4,974.76	3
LB is not used	20	3,012.34	$1.29 * 10^8$	8,112.25	25
	30	Too long	Unavailable	Unavailable	30
All features are used	20	3.67	89,135.34	148.34	0
	30	78.34	$5.84 * 10^6$	2,639.67	0

The efficiencies of upper bound, dominance criteria, and lower bound are summarized in Table 3.8. The results indicated that the lower bound is the most important feature of the algorithm. If the lower bound does not exist, branch and bound algorithm cannot efficiently solve even small sized problems. The dominance criteria are the second most important feature of the algorithm. The exclusion of dominance criteria results in a considerable increase in the computation time and the number of generated nodes for both problem sizes. The upper bound is the least important feature. However, for relatively large problems (No. of jobs = 30), the effect of upper bound is still significant.

The upper bound derived from the two-phase heuristic procedures is the least important feature because it is only useful at the early stage of the computation. It is applied to obtain the upper bound since the initialization step of the algorithm. If the upper bound is not calculated at the initialization step, the equivalent or better one can still be obtained after the algorithm has found a sufficiently good incumbent solution and has spent some computational time. However, the good upper bound can save this amount of computational time and allow a number of nodes to be eliminated at the beginning of the search tree, especially for large-sized problem.

On the other hand, the lower bound and dominance criteria are repeatedly calculated at every node, so they are capable of eliminating a larger number of nodes than the upper bound. Note that the upper bound derived from the two-phase heuristic procedures is effective only at the early stage of the computation while the lower bound

and dominance criteria are effective for all stages of computation. The lower bound calculates the least cost that will incur or the best objective value of that node if it is branched further. When adding the lower bound to the associated cost of the particular node, there is a good chance that the cost of branching further will be greater than the current upper bound and the node will be eliminated. The dominance criteria consider only the precedence relations of the adjacent and non-adjacent jobs within the node (the scheduled jobs). They do not consider the effect of other jobs which are still not included in the sequence (the unscheduled jobs). Therefore, the lower bound has a greater effect on the performance of the algorithm than the dominance criteria.

3.6 Concluding Remark

In this chapter, two types of branch and bound algorithm, B&B1 and B&B2, are developed to determine a global optimal production schedule on a single machine that minimizes sequence dependent setup cost and earliness/tardiness penalties when the setup time is also sequence dependent. A number of single machine scheduling problems found in literature are special cases of the problem under consideration. Therefore, the proposed scheduling method can be applied to solve these special cases by modifying some input parameters. The computational experiment reported that B&B2 is more efficient than B&B1. The B&B 2 can be used to solve the problem with up to 40 jobs while B&B1 can solve only 16 jobs problem. This indicated that lower bound, upper bound, and dominance criteria are very efficient in eliminating numbers of nodes. The results indicated that lower bound is the most significant feature of the algorithm.