Supported by the National Science Foundation Engineering Research Center Program (NSFD CD 8803012), Industry and the University

Single Machine Scheduling with Discrete Earliness and Tardiness

by G. Harhalakis, R. Nagi and J.M. Proth

TR 92-79

Single Machine Scheduling with Discrete Earliness and Tardiness

G. Harhalakis[1*], R. Nagi[1], J.M. Proth[2]

[1]Department of Mechanical Engineering & Systems Research Center,
University of Maryland, College Park, MD 20742, U.S.A.
*Senior Member IIE
[2]INRIA, 4, rue Marconi, Technopole Metz 2000, 57070 METZ, France,
& Systems Research Center, University of Maryland, U.S.A.

This paper considers the problem of scheduling a given set of jobs on a single machine in order to minimize the total weighted earliness and tardiness costs. The scheduling horizon is divided into elementary periods; jobs have due-dates at the end of these periods. All jobs are assumed initially available. Jobs have unique (weighted) early and tardy staircase penalty functions. No preemption of jobs is permitted, and idle time may be inserted. We prove that this problem is NP-complete. Some results relating to job priorities and completion times in an optimal solution are presented. A Mixed Integer Linear Programming (MILP) formulation of this problem is developed. A branch-and-bound scheme that solves the above problem optimally is also presented. Heuristics, derived from simple priority rules, provide an initial upperbound to the search. We develop two lower bound procedures for the remaining jobs to be scheduled at any partial solution state. Numerical results relating to the performance of the branch-and-bound scheme are also presented.

Key words : Production/Scheduling, Deterministic Sequencing, Single Machine Sequencing, Early/Tardy Problem.

1 Introduction

Production scheduling problems have been recognized for their complexity. Many scheduling problems with varying criteria, assumptions relating to release and due dates, and number of production stages have been treated in the literature. Among these, the single stage (machine) scheduling has been extensively studied. French (1982), and Conway *et al* (1967) present some criteria of interest, while Gupta and Kyparisis (1987) review the literature related to this problem. Typical criteria include : mean tardiness, mean lateness, total weighted tardiness, maximum tardiness, number of tardy jobs, weighted number of tardy jobs, etc. Previously, the research focus was on *regular* measures. Regular measures are nondecreasing with respect to job completion time, i.e., given two schedules where completion times of jobs in the first schedule are no later than the ones in the second schedule, then under a regular measure, the first schedule is at least as good as the second one. More recently, with the advent of the Just-In-Time (JIT) concept, research attention has been drawn towards combined penalization of earliness as well as tardiness. This gives rise to a *nonregular* performance measure.

Furthermore, most of the scheduling literature considers that due-dates and delivery times of jobs can assume continuous values on a time-horizon. Consequently, the earliness and tardiness also assume continuous values. Models with such characteristics, although applicable to a good variety of practical scheduling problems, fail to capture an important class of problems encountered in business practice. Consider for instance a business with periodic shipment, e.g. the delivery truck leaves once a week. Failure of a job to be completed by the due delivery time results in waiting for the next shipment, regardless of the actual tardiness (expressed in a continuous manner). On the other hand, the inventory book-keeping function can also be periodic, where interest on capital is accrued at the end of a business day. Thus, a part taken into stock is penalized (by a fixed amount) at the end of the day, regardless of the actual completion time during the day. Similar examples can be found in a make-to-forecast environment, where due-dates are assigned at the end of a reasonable forecasting sub-period (e.g. week) rather than in a continuous fashion.

In reference to such issues, we consider the problem of scheduling a set of initially available jobs, to be produced within a given scheduling horizon (e.g. week), on a single machine, with the objective to minimize the total earliness and tardiness penalties, subject to no preemption constraint. The scheduling horizon is discretized into elementary periods; *jobs are due, and can be delivered only at the end of these periods* (e.g. day). The penalty functions are discrete and weighted (staircase). Set-up times are not accounted for, and a job can be processed in two elementary periods if necessary. This problem differs from previous work in this area in that the earliness and tardiness penalty functions are discontinuous, as opposed to being continuous (linear or quadratic). A job, completed in a period prior to that of its due date, is penalized by the number of periods (an integer) it spends in inventory, weighted appropriately by its holding cost per period. Analogously, when a job is tardy, it is penalized by the number of periods (an integer) by which it is delayed, weighted appropriately by its backlogging cost per period.

It is important to indicate that this does not emerge as a minor variation of the continuous-time version of the single machine Earliness and Tardiness (E/T) scheduling problem. The solution to the continuous-time problem may provide very poor results for our version of the problem. This is because the continuous-time problem penalizes a due-date violation by the parts' weight multiplied by the duration of E/T; if the E/T tends to zero, the penalty tends to zero. However, in our version, penalization has jumps, and even a small tardiness violation may cause significant cost. Thus, one can appreciate that the characteristics of optimal (or good) solutions to the two versions of the problem may differ significantly.

Our version of the problem does not seem to have drawn much attention in the literature, while the continuous-time version has been extensively studied. Most of the previous work in single-stage scheduling dealt with single performance measures (see Baker 1974), especially tardiness. Gupta and Kyparisis (1987), and Sen and Gupta (1984) review literature related to variations of this problem. Sidney (1977), and Lakshminarayan *et al* (1978) present algorithms for minimizing the maximum penalty of early or late jobs. A comprehensive survey relating to E/T models can be found in Baker and Scudder (1990). They cite several publications with different assumptions like: common due-dates, unweighted or equal E/T weights, no insertion of idle time, etc. We restrict this discussion to the continuous-time, distinct due-date, weighted total E/T model. Garey *et al* (1988) prove that the unweighted problem is NP-complete. Hall and Posner (1991) prove that the common due-date problem is NP-complete. Abdul-Razaq and Potts (1988) consider this problem without inserted idle time. They employ a branch-and-bound scheme using dynamic programming state-space relaxation techniques for obtaining lower bounds. Their computation results suggest that problems with more than 20 jobs may lead to excessive solution times. Ow and Morton (1989) also consider the problem without inserted idle time. They suggest dispatch heuristics, and propose a Filtered Beam Search method. Fry *et al* (1987), and Garey *et al* (1988) develop efficient procedures of optimally inserting idle time for a given job sequence. Fry *et al* (1987) decompose the problem by determining a good sequence, followed by the optimal insertion of idle time. Ferris and Vlach (1992) consider the general E/T problem, and show that certain criteria are tractable polynomially. We are not aware of any literature related to periodic distinct due-date weighted total E/T problem with staircase penalty functions.

Owing to the nonregular nature of our objective function (i.e., that can decrease with increasing completion times), important theorems (Emmons 1969, Lawler 1977, Smith 1956, etc.) on job ordering cannot be used as pruning devices for Branch and Bound or Dynamic Programming algorithms. Furthermore, the insertion of idle time makes the problem even more complex. The discrete penalty functions, however, render the ordering of jobs within a period unimportant. Thus, the problem reduces to the determination of the period in which each job is completed.

We formulate this problem as a Mixed Integer Linear Programming (MILP) Problem, and present some results regarding job priorities and completion times in an optimal schedule. We then develop a Branch-and-Bound scheme to obtain optimal results for the above stated problem.

In section 2, the problem formulation is presented, detailing first the manufacturing system, the demand, and the criteria of interest. The proof of NP-completeness is presented in section 3. In section 4, some preliminary results relating to job priorities and completion times in an optimal solution are presented. A Mixed Integer Linear Programing (MILP) formulation of the problem is presented in section 5. In section 6, we describe the Branch-and-Bound scheme that solves the problem optimally. Section 7 is devoted to some numerical examples. Finally, in section 8 we draw our conclusions and present our recommendations for future work.

2 Problem Formulation

In this section, we formally describe the details and notations of the problem at hand, and present its formulation.

We consider a single-stage manufacturing system consisting of a single machine M. Let H denote the scheduling horizon which is composed of z elementary periods. An elementary period is any convenient period, e.g. a day; it has a duration of Δ time units. Although it is not restrictive that the duration of the elementary periods be equal (with duration of Δ), we assume this for simplicity. We consider the problem of scheduling a total number of N jobs. Job i is described by : (1) t_i, $t_i \in \mathrm{IR}^+$ (IR^+ is the set of positive real numbers), denoting the processing time, and (2) d_i, $d_i \in \{1,2,\ldots,z\}$, denoting the due date. It is assumed that $t_i <$

Δ; i = 1,2,...,N. We associate with each job i, an earliness weight w_i^+, and a tardiness weight w_i^-, when i is completed one period before or after its due date, respectively ($w_i^+, w_i^- \in \mathrm{IR}^+$). Let C_i, $C_i \in \mathrm{IR}^+$, denote the completion time of job i. Then, x_i, $x_i \in \{1,2,\ldots,z, z+1\}$, denoting the elementary period in which job i is completed, can be computed as follows :

$$x_i = \begin{cases} \lceil C_i/\Delta \rceil & \text{if } C_i < (z+1)\,\Delta \\ z+1 & \text{otherwise} \end{cases} \quad ; i = 1,2,\ldots,N \tag{2.1}$$

where $\lceil \bullet \rceil$ denotes the smallest integer greater than or equal to $\bullet$. This implies that the completion of a job outside the scheduling horizon H is assumed to occur in period (z+1), or, the duration of that last elementary period (z+1) is very large. Let E_i and T_i represent the discrete earliness and tardiness, respectively, of job i; these are detailed as :

$$E_i = \mathrm{Max}(d_i - x_i, 0) = (d_i - x_i)^+;\ i = 1,2,\ldots,N, \text{ and} \tag{2.2}$$

$$T_i = \mathrm{Max}(x_i - d_i, 0) = (x_i - d_i)^+;\ i = 1,2,\ldots,N. \tag{2.3}$$

Note : $E_i, T_i \in \mathrm{IN}$. Thus, the penalty function versus the completion time of job i with a due-date k is presented in figure 1.

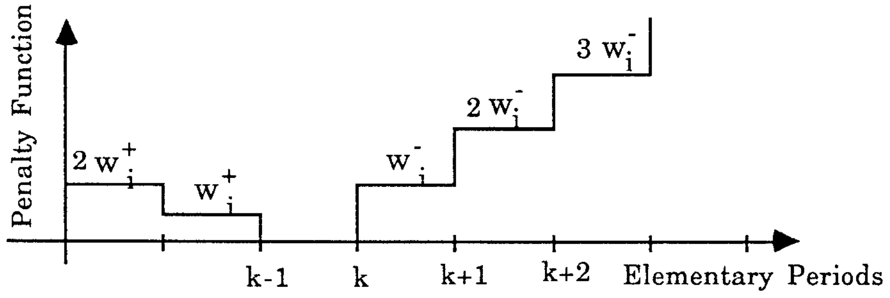

Figure 1 : Penalty function of job i with due-date k

Let S represent a schedule. The scheduling problem can be formally stated as follows (*P1*):

Minimize : $$\mathrm{Cost}(S) = \sum_{i=1}^{N} (w_i^+ \times E_i + w_i^- \times T_i) \tag{2.4}$$

Subject to : $$[C_r - t_r, C_r) \cap [C_s - t_s, C_s) = \varnothing;\quad r, s = 1,2,\ldots,N, \text{ and } r \neq s. \tag{2.5}$$

$$C_i - t_i \geq 0;\qquad i = 1,2,\ldots,N \tag{2.6}$$

(2.1), (2.2), and (2.3).

Constraint (2.5) signifies that two jobs cannot be processed at the same time, (2.6) implies that jobs cannot have negative starting times. The criteria to be minimized (2.4), represents the total weighted discrete earliness and tardiness penalties. These criteria combined result in a nonregular performance measure. In our formulation, we are not restricted by assumptions such as *"agreeable"* weighting of jobs (i.e., $t_i < t_j$ implies $w_i^- \geq w_i^-$) (Lawler 1977); backlogging costs, inventory costs, and processing times need not hold any specific relationship.

This problem can be generalized to periods of unequal duration; let Δ_k represent the duration of period k, for $k = 1,2,\ldots,z$, and $\Delta_{z+1} = \infty$. Then, the time at which the k-th period ends is defined as ∂_k, and is computed as follows: $\partial_1 = \Delta_1$, and $\partial_k = \sum_{j=1}^{k} \Delta_j$, for $k = 2,3,\ldots,z+1$.

The general formulation has practical significance, for instance, when a company does not operate the same number of shifts everyday. In this case, the formulation ($\mathcal{P}1$) remains unchanged, except for a new definition of completion periods (i.e. 2.1):

$$x_i = k \qquad \text{if } C_i \in [\partial_{k-1}, \partial_k); \qquad i = 1,2,\ldots,N \tag{2.1'}$$

The problem can be further generalized to period specific penalties, where each job has a specific penalty for completion in a given period; thus, the penalty function is not a regular step function as presented in Figure 1. Such a generalization could capture variable cost discounting.

3 NP-completeness

As mentioned earlier, the weighted total E/T model with continuous E/T penalty functions is NP-complete (Garey *et al*, 1988). We are not aware that the problem under consideration (with staircase E/T penalty functions) has been addressed or proven NP-complete in the literature. In this section, we demonstrate that the scheduling problem ($\mathcal{P}1$) is NP-complete, hence it is unlikely to be solved by a polynomial time algorithm. More precisely, we show that the decision problem, $\mathcal{D}$, "Is there a schedule with total cost no more than a chosen value F ?," is NP-complete; hence the optimization problem is at least as hard.

(1) The scheduling problem is in NP.
Proof : It is easily seen to be in NP as a nondeterministic algorithm for it need only determine the value of x_i (nondeterministic polynomial time), and check in polynomial time if the schedule costs no more than F.

(2) The scheduling problem is in NP-complete.
Proof by Restriction : We restrict $\mathcal{D}$ to solve the KNAPSACK problem (Garey and Johnson 1979), which is a well known NP-complete problem.
KNAPSACK problem instance : A finite set U, a "size" $s(u) \in Z^+$ and a "value" $v(u) \in Z^+$ for each $u \in U$, a size constraint $B \in Z^+$, and a value goal $K \in Z^+$.

Decision problem : Is there a subset U' of U such that :

$$\sum_{u \in U'} s(u) \leq B \quad \text{and} \quad \sum_{u \in U'} v(u) \geq K \tag{3.1}$$

Restrictions on $\mathcal{D}$:

(1) Let number of periods, $z = 1$.

(2) Let duration of first period, $\Delta_1 = B$ ($\Delta_2 = \infty$).

(3) $\forall u_i \in U$, let $t_i = s(u_i)$, $d_i = 1$, $w_i^- = v(u_i)$ and $w_i^+ = 0$.

(4) If $F = \sum_{u \in U} v(u) - K$

It can now be seen that the answer to the decision problem $\mathcal{D}$, is in fact the answer to the Knapsack problem (3.1). Since the Knapsack problem is NP-complete, it implies that the decision problem $\mathcal{D}$ is also NP-complete.

Q.E.D.

4 Preliminary Results

Theorem 1:

For two jobs i and j, such that $t_i = t_j$, $d_i \geq d_j$, $w_i^+ \geq w_j^+$, and $w_i^- \leq w_j^-$ we obtain $x_i \geq x_j$ (j precedes i) in an optimal schedule.

Proof :

We use the standard interchange argument (Garey *et al*, 1988). Since $t_i = t_j$, i and j can be interchanged to provide a feasible schedule without changing the completion times of the remaining jobs, i.e., the cost of the remaining schedule. Let $C_i < C_j$, with $x_i \leq x_j$ (i precedes j) in a schedule S. If $x_i = x_j$, the interchange will retain $x^*_i = x^*_j$ (although $C^*_i > C^*_j$) leaving the cost of the new schedule unchanged. We consider the case $x_i < x_j$. There are six cases depending on the relative ordering of d_i, d_j, x_i, and x_j:

(1) For $d_i \geq d_j \geq x_j > x_i$, the interchange results in $d_i \geq d_j \geq x^*_i > x^*_j$, and a resulting cost reduction of $(w_i^+ - w_j^+)\times(x^*_i - x^*_j)$; note: $(w_i^+ - w_j^+) \geq 0$ and $(x^*_i - x^*_j) > 0$,

(2) For $d_i \geq x_j > d_j \geq x_i$, the interchange results in $d_i \geq x^*_i > d_j \geq x^*_j$, and a resulting cost reduction of $w_i^+\times(x^*_i - x^*_j) + w_j^-\times(x^*_i - d_j) - w_j^+\times(d_j - x^*_j)$; note: (i) $(x^*_i - x^*_j) > (d_j - x^*_j) \geq 0$ and $w_i^+ \geq w_j^+$, which gives $w_i^+\times(x^*_i - x^*_j) - w_j^+\times(d_j - x^*_j) \geq 0$, and (ii) $(x^*_i - d_j) > 0$,

(3) For $d_i \geq x_j > x_i \geq d_j$, the interchange results in $d_i \geq x^*_i > x^*_j \geq d_j$, and a resulting cost reduction of $(w_i^+ + w_j^-)\times(x^*_i - x^*_j)$,

(4) For $x_j > d_i \geq d_j > x_i$, the interchange results in $x^*_i > d_i \geq d_j > x^*_j$, and a resulting cost reduction of $(w_j^-\times(x^*_i - d_j) - w_i^-\times(x^*_i - d_i)) + (w_i^+\times(d_i - x^*_j) - w_j^+\times(d_j - x^*_j))$; note: (i) $(x^*_i - d_j) \geq (x^*_i - d_i) > 0$ and $w_j^- \geq w_i^-$, which gives $(w_j^-\times(x^*_i - d_j) - w_i^-\times(x^*_i - d_i)) \geq 0$, and (ii) $(d_i - x^*_j) \geq (d_j - x^*_j) > 0$, and $w_i^+ \geq w_j^+$, which gives $(w_i^+\times(d_i - x^*_j) - w_j^+\times(d_j - x^*_j)) \geq 0$,

(5) For $x_j > d_i \geq x_i \geq d_j$, the interchange results in $x^*_i > d_i \geq x^*_j \geq d_j$, and a resulting cost reduction of $w_j^- \times (x^*_i - x^*_j) + w_i^+ \times (d_i - x^*_j) - w_i^- \times (x^*_i - d_i)$; note: (i) $(x^*_i - x^*_j) \geq (x^*_i - d_i) > 0$ and $w_j^- \geq w_i^-$, which gives $w_j^- \times (x^*_i - x^*_j) - w_i^- \times (x^*_i - d_i) \geq 0$, and (ii) $(d_i - x^*_j) \geq 0$, and

(6) For $x_j > x_i \geq d_i \geq d_j$, the interchange results in $x^*_i > x^*_j \geq d_i \geq d_j$, and a resulting cost reduction of $(w_j^- - w_i^-) \times (x^*_i - x^*_j)$; note: $(w_j^- - w_i^-) \geq 0$.

Therefore, the interchange of i and j, creates a new schedule with a value of the criterion at least as good as the one of the previous schedule.

Q.E.D.

By employing this argument of interchangeability repetitively, we can define a partial ordering on the jobs. For industrial problems this result implies that jobs that belong to the same part type can be ordered according to EDD (Earliest Due-Date) with ties broken arbitrarily. Hereafter, we refer to the number of part types by n and the number of jobs by N; obviously, for a problem instance $n \leq N$.

Theorem 2:

An optimal schedule can be constructed by inserting no idle time between jobs that have completion times within the same elementary period.

Proof :

From an optimal schedule, characterized by C^*_i, for i = 1,2,...,N, we can determine completion periods x^*_i, for i = 1,2,...,N. Any schedule in which the job completion periods are x^*_i, for i = 1,2,...,N, is also an optimal schedule; hence, the result holds trivially for the above case (see Algorithm 1).

Algorithm 1:

Given a schedule partially defined by x_i, for i = 1,2,...,N, we present here a linear time algorithm (O(N)) for determining the job completion times, C_i, for i = 1,2,...,N, i.e., inserting idle time.

For each elementary period k, let L_k represent the job with the longest processing time that is completed in k; k = 1,2,...,z. That is, $L(k) = \underset{i}{\operatorname{argmax}}\{t_i \mid x_i = k\}$; this takes O(N) time.

Now, we define an ordering for the jobs as follows. For each elementary period k (k = 1,2,...,z), the job with the longest processing time, L(k), is sequenced the first; the other jobs to complete in this period are sequenced arbitrarily (or according to the job index) after it. The completion times are determined by inserting no idle time between jobs that have completion times within the same elementary period. The algorithm is presented as follows.

Notation :
n(k) = number of jobs with completion time within the k-th elementary period.
I(p,k) = index of the p-th job of the k-th elementary period; Note I(1,k) = L(k); k = 1,2,...,z.
These arrays can be created in linear time while determining the longest jobs and the ordering.

```
time := 0;
for k = 1 to z do
        for p = 1 to n(k) do
                $C_{I(p,k)}$ := time + $t_{I(p,k)}$;
                time := time + $t_{I(p,k)}$;
        end;
        time := Max(time, $\partial_k + \varepsilon - t_{I(1,k+1)}$);     /*   Earliest start time of job I(1,k+1)   */
end.
```

ε is an infinitesimally small real positive number.

Thus, given an optimal schedule partially characterized by x^*_i, for i = 1,2,...,N, the idle time can be inserted by Algorithm 1; i.e., C^*_i, for i = 1,2,...,N, can be determined.

This algorithm can also be employed for checking in linear time the feasibility of a schedule characterized by x_i, for i belonging to any subset of {1,2,...,N}. This can be achieved by introducing the following check after the end of the inner loop.

```
        if(time > k $\Delta$)
                print("Infeasible Schedule");
        else
                time := Max(time, $\partial_k + \varepsilon - t_{I(1,k+1)}$);
```

5 A Mixed Integer Linear Programming Formulation

Based on the remarks in the previous section, we present a Mixed Integer Linear Programing (MILP) formulation of the problem $\mathcal{P}1$. The major reason of presenting this is the relatively small number of 0-1 variables required in this case (N(z+1)) compared to that of most other scheduling problems ($O(N^2)$), under the reasonable assumption that N >> z (refer to assumption $t_i < \Delta_\bullet$; i = 1,2,...,N in section 2). Needless to mention that the applicability of the MILP approach remains limited to problems of small dimension. Furthermore, we first present the formulation without precedence relations specified by Theorem 1.

We define N(z+1) number of 0-1 variables, that indicate the completion periods of jobs.

$$J(i,k) = \begin{cases} 1 & \text{if } x_i = k \\ 0 & \text{otherwise} \end{cases} ; i = 1,2,\ldots,N, k = 1,2,\ldots,z+1 \tag{5.1}$$

Let z1 = z+1.

Minimize : $$\text{Cost}(S) = \sum_{i=1}^{N} P_i \tag{5.2}$$

Subject to : $$P_i \geq w_i^+ \times (d_i - (\sum_{k=1}^{z1} k \times J(i,k))); \quad i = 1,2,\ldots,N \tag{5.3}$$

$$P_i \geq w_i^- \times ((\sum_{k=1}^{z1} k \times J(i,k)) - d_i); \quad i = 1,2,\ldots,N \tag{5.4}$$

$$\sum_{k=1}^{z1} J(i,k) = 1; \quad i = 1,2,\ldots,N \tag{5.5}$$

$$B_k \geq t_i \times J(i,k); \quad i = 1,2,\ldots,N, k = 1,2,\ldots,z \tag{5.6}$$

$$(\sum_{i=1}^{N} t_i \times J(i,k)) - B_k \leq \Delta_k - \varepsilon; \quad k = 1,2,\ldots,z \tag{5.7}$$

$$\sum_{j=1}^{k} \sum_{i=1}^{N} (t_i \times J(i,j)) \leq \partial_k; \quad k = 1,2,\ldots,z \tag{5.8}$$

$$P_i \geq 0 \ (i = 1,2,\ldots,N); B_k \geq 0 \ (k = 1,2,\ldots,z); J(i,k) \in (0,1) \ (i = 1,2,\ldots,N, k = 1,2,\ldots,z1)$$

P_i represents the penalty cost of job i; constraints (5.3) and (5.4) ensure that the appropriate penalty among earliness or tardiness is taken up. Constraint (5.5) ensures that each job is completed in only one period. B_k represents the processing time of the longest job completing in period k; constraint (5.6) determines this. Constraint (5.7) ensures that completion of the jobs assigned to a period is feasible within the duration of that elementary period, while (5.8) ensures that the jobs completing in adjacent periods do not overlap.

Precedence relations among jobs according to theorem 1 can be imposed in the form of the following constraints. For instance, if job r precedes job s (r, s $\in$ (1,2,...,N), r≠s) in an optimal schedule, we have:

$$\sum_{k=1}^{z1} k \times J(r,k) \leq \sum_{k=1}^{z1} k \times J(s,k); \tag{5.9}$$

6 A Branch and Bound Algorithm

As mentioned in the previous section, the applicability of the MILP approach remains limited to problems of small dimension. In this section, we present a Branch and Bound algorithm that solves the scheduling problem more efficiently. The general principle behind the algorithm is to attempt to improve upon a "good" feasible solution, until there are no solutions of lower cost.

6.1 Initial upper bound : Heuristic rules

In this section, we describe four heuristic scheduling rules in order to obtain a feasible schedule with a low cost. These rules essentially prioritize the jobs; jobs with higher priority are scheduled in the least expensive "location" first.

Rule 1 : Jobs are arranged in nonincreasing order of the ratio $(w_i^- + w_i^+)/t_i$.
Rule 2 : Jobs are arranged in nonincreasing order of $(w_i^- + w_i^+)$.
Rule 3 : Jobs are arranged in nonincreasing order of the ratio $\text{Max}(w_i^-, w_i^+)/t_i$.
Rule 4 : Jobs are arranged in nonincreasing order of $\text{Max}(w_i^-, w_i^+)$.

At each step, the heuristic selects the first unscheduled job and schedules it in the least expensive "location". The least expensive "location" is defined as the one that contributes the least to the cost function. In other words, if feasible, schedule the completion of a job in the period corresponding to its due date, otherwise attempt to schedule its completion in the period before (after) its due date, if the earliness penalty is less than (greater than) the tardiness penalty. If the job remains unscheduled, schedule its completion in a first feasible period before or after its due date that imposes the least penalty. Feasibility of a particular completion assignment can be verifyed by algorithm 1.

Several trials were performed with varied number of part types (n), number of jobs (N), number of periods (z), penalties, processing times and due-dates. In Table 6.1, we present the percentage of problems for which that rule provided the lowest cost. $\overline{N}$ represents the average number of jobs over 270 or 405 problem instances (Runs). As observed from these results, in general, it is difficult to claim that one rule outperforms the others. Table 6.1 also indicates the average percentage error of the best rule from the optimum. In about 50% of the cases, one of the heuristics was able to provide a bound within 10% of the optimum. The detailed description of the problem instances is presented in section 7.

Note that the sum is greater than 100 because for some cases more than one rule provided the lowest cost.

Problem Instances				Rule 1	Rule 2	Rule 3	Rule 4	Avg. % Error from Opt.
n	$\overline{N}$	z	Runs					
2	9.13	3	270	95%	84%	95%	84%	18%
2	14.23	5	405	94%	85%	94%	85%	21%
4	9.45	3	270	82%	77%	78%	63%	38%
4	14.94	5	405	75%	76%	70%	57%	36%
6	9.42	3	270	73%	82%	73%	76%	35%
6	14.7	5	405	74%	82%	72%	67%	27%

Table 6.1 : Number of times (percentage) bound found by each rule

6.2 The Detailed Branch and Bound Algorithm

The algorithm consists of 3 major phases: initialization, branching and termination.

Initialization consists of the determination of: (i) an initial upper bound, (ii) an initial sequence of jobs to be considered during the branching phase, (iii) job priorities based on theorem 1, and (iv) an initial lower bound.

Branching is an iterative procedure for developing the search tree. A node in the search tree represents a partial schedule (assignment of completion periods, x_i's, to a set of jobs). A node is developed or expanded by generating descendent nodes. A descendent node represents the inclusion of an additional unscheduled job to the partial schedule of its parent node. At each iteration, the procedure: (1) identifies an unmarked node to be expanded (a depth-first strategy is employed for various reasons; see last paragraph in this sub-section; hence, the node under expansion is the one with the maximal number of scheduled jobs, with ties broken according to lower bound cost), (2) generates descendents of the current node, (3) eliminates dominated nodes, (4) computes lower bounds of the non-eliminated nodes and updates the list of active (unmarked) nodes; the current node is then marked. An unmarked node is dominated if its lower bound is not strictly smaller than the current best upper bound (best feasible schedule so far).

The algorithm stops when the list of active nodes is empty. The termination phase consists of identifying the current best upper bound as the optimal solution, and computing the completion times of jobs for this schedule (using algorithm 1 of section 4).

Hence, during the search, we keep the current best feasible schedule (upper bound), S, whose cost is denoted by UB, and a list of unmarked nodes, L. The list L is ordered according to the sequence of nodes generated; ties are broken by higher 'lower bound cost' first, for the nodes generated at the same branching iteration. Thus, nodes representing a higher number of scheduled jobs (longest partial-schedules, or deepest nodes of the search tree) are towards the head of the list, with the ones of lower 'lower bound cost' towards the head of the list for the nodes at the same depth. The marked nodes represent partial (or complete) schedules for the unmarked nodes (or best schedule S). Each marked node o represents the following information :
J(o): Job number
X(o): Completion period
P(o): Pointer to parent marked node (having similar information); NIL for the first job.

The unmarked nodes are maintained in the list L. Each unmarked node • represents the following information :
j(•): Job number
x(•): Completion period
p(•): Pointer to parent marked node (representing a partial schedule; see note 1 below)
c(•): Cost of the partial schedule (including j(•))
e(•): Sum of the lower bound cost of remaining jobs and cost of the partial schedule
l(•): Pointer to the next unmarked node of the list L.

Note 1: We also define Y(~,•) as the completion period of a job ~ in the partial schedule represented by p(•). Thus, Y(i,•) for i = 1,2,...,j(•)-1 represents the partial schedule obtainable from p(•). Further, let Y(0,•) = 1.

Memory location for unmarked nodes and their partial schedules are freed when they are marked. And each of the descendent nodes copies the information of the new partial schedule, which is the partial schedule of the parent node being marked along with its completion period. This representation, where each unmarked node carries its partial schedule, was preferred over the one where the marked nodes are retained in the memory in a tree structure, and each unmarked node simply points to a node in the tree because of the following reasons. The number of marked nodes grows substantially during execution and they have to be retained in the system at all time, while the maximal number of unmarked nodes is not very high. Thus, it presents lower memory requirements.

6.2.1 Initialization

In the first step of the initialization, we employ the four heuristic rules detailed in section 6.1 to obtain feasible solutions of the scheduling problem; the schedule S that provides the lowest cost serves as the initial upper bound (UB). The initial sequence (IS) of jobs to be considered during the branching phase is obtained by ordering jobs according to earliest due date first, breaking ties with lower job number first. The job priorities are computed on the basis of theorem 1. In fact, we retain for each job in IS only one job (if any), with the maximal index, which precedes it in an optimal schedule. Let PR(i) represent the job that precedes job i, with $PR(i) < i$, and $PR(i) = 0$ if no job precedes job i in accordance with theorem 1. Finally, a lower bound for scheduling the entire set of jobs is computed according to the procedure detailed in section 6.3.

Initially, the list of unmarked nodes (L), and the partial schedules of marked nodes are empty; i.e. they are null pointers.

6.2.2 Branching

The algorithm tries to expand a node μ at the head of the list L, except for the first iteration, where it tries to expand the first job in the sequence of unscheduled jobs. Expansion of an unmarked node μ is performed only if $e(\mu) < UB$, and consists of generating the children nodes and placing the appropriate ones in the list L. The node μ is then marked. For node μ, we attempt to generate a maximum of (z+1) nodes, representing the completion of the next unscheduled job, $(j(\mu) + 1)$, in one of the elementary periods. That is, we can generate node $ç_i$, for $i = 1,2,\ldots,(z+1)$; $j(ç_i) = (j(\mu) + 1)$, $x(ç_i) = i$, $p(ç_i) = \mu$, for $i = 1,2,\ldots,(z+1)$. The child nodes that are appropriate to be added to L are determined based on the following rules.

Test 1: Node $ç_i$ is added only if $x(ç_i) \geq Y(PR(j(ç_i)),ç_i)$. That is, the current job cannot be scheduled in a period prior to the completion period of the job that precedes it in accordance with theorem 1.

Test 2: Node $ç_i$ is added only if the partial schedule $Y(\pi,ç_i)$, for $\pi = 1,2,\ldots,j(ç_i)-1$, and $x(ç_i)$ for $j(ç_i)$ is feasible; the feasibility is ascertained by algorithm 1.

Test 3: Node $ç_i$ is added only if $e(ç_i) < UB$. That is, the sum of the lower bound cost of remaining jobs and cost of the partial schedule is strictly less than the current upper bound. The computation of the lower bound cost for remaining jobs is presented in section 6.3.

The set of nodes that pass tests 1 through 3 are added to the list L after ordering them; nodes with higher $e(ç_i)$ are added first. The memory corresponding to parent node μ is freed.

In addition, if $j(ç_i) = N$, i.e. if the children nodes (that verify tests 1-3) represent scheduling of the last job, we have the following updates : (i) the upper bound UB is updated as $\text{Min}\{e(ç_i)\}$, and (ii) the best schedule S is updated by $p(ç_i)$ and $x(ç_i)$.

6.2.3 Termination

The branch and bound algorithm ends when the list L becomes empty. Then, S represents the optimal schedule, and UB represents the total cost for the optimal schedule.

There is another type of termination, when the algorithm is unable to successfully terminate in a prespecified execution time. In this case, S is chosen as the best schedule obtainable in the specified execution time, and UB represents its cost. However, in this case, we cannot be certain about the optimality of this schedule.

6.3 Lower Bound for Remaining Jobs

In this section, we present the lower bound for scheduling remaining jobs for a given partial schedule. We employ a "boundary-relaxation" approach for computing this bound. We consider each elementary period separately. For each period k, we determine the set of unscheduled jobs with a due date at the end of period k, i.e., $\{i \mid d_i = k$, and is unscheduled$\}$. We determine the optimal (lowest) cost of scheduling these jobs in period k, given the set of jobs that are previously scheduled for completion in k, i.e. given $\{j \mid x_j = k$, and is scheduled$\}$, with unconstrained boundary conditions from the neighboring period(s). Boundary conditions of zero time (beginning of the scheduling horizon) are still respected by the first elementary period. For the other elementary periods, relaxed boundary conditions can be considered as two periods of infinite duration on either side of the elementary period. However, theorem 3 enables us to convert this problem into an equivalent problem with only one period of infinite duration before the elementary period.

<u>Theorem 3:</u>
Consider a period of duration Δ_2 which is bordered on each side by periods of very large duration (larger than the total job processing times; thus, a total of three periods with $\Delta_1 = \Delta_3$ = a very large number, V). The problem of scheduling a given set of jobs having $d_i = 2$, $\forall i$, with earliness and tardiness, is equivalent to a scheduling problem with only earliness in which the earliness weight for each job i is defined as $\text{Min}\{w_i^-, w_i^+\}$; $i = 1,2,\ldots,n$.

Proof :

We consider the MILP formulation of this problem as presented in section 5.

We define 3 n number of 0-1 variables, indicating the completion periods of jobs.

$$J(i,k) = \begin{cases} 1 & \text{if } x_i = k \\ 0 & \text{otherwise} \end{cases} ; i = 1,2,\ldots,n,\ k = 1,2,3 \qquad (6.1)$$

Minimize : $$\text{Cost}(S) = \sum_{i=1}^{n} w_i^{+} \times J(i,1) + w_i^{-} \times J(i,3) \qquad (6.2)$$

Subject to : $$\sum_{k=1}^{3} J(i,k) = 1; \qquad i = 1,2,\ldots,n \qquad (6.3)$$

$$B_2 \geq t_i \times J(i,2); \qquad i = 1,2,\ldots,n \qquad (6.4)$$

$$\left(\sum_{i=1}^{n} t_i \times J(i,2)\right) - B_2 \leq \Delta_2 - \varepsilon; \qquad (6.5)$$

$$P_i \geq 0\ (i = 1,2,\ldots,n);\ B_2 \geq 0;\ J(i,k) \in (0,1)\ (i = 1,2,\ldots,n,\ k = 1,2,3) \qquad (6.7)$$

Note that constraint of type (5.8) is irrelevant in this case owing to the very large duration of boundary periods.

We transform the above problem as follows :

Minimize : $$\text{Cost}(S) = \sum_{i=1}^{n} \text{Min}\{w_i^{+}, w_i^{-}\} \times (J(i,1) + J(i,3)) + ET_i \qquad (6.8)$$

Subject to : $$ET_i \geq (w_i^{+} - w_i^{-}) \times J(i,1); \qquad i = 1,2,\ldots,n \qquad (6.9)$$

$$ET_i \geq (w_i^{-} - w_i^{+}) \times J(i,3); \qquad i = 1,2,\ldots,n \qquad (6.10)$$

$$ET_i \geq 0; \qquad i = 1,2,\ldots,n \qquad (6.11)$$

(6.3) - (6.7)

Now it can be observed that $ET_i = 0$, for $i = 1,2,\ldots,n$, in an optimal solution of the above problem. In other words, in an optimal solution if job i is early (resp. tardy), it implies that $w_i^{+} \leq w_i^{-}$ (resp. $w_i^{+} \geq w_i^{-}$). Otherwise, an alternative solution of lower cost can be constructed by switching the job i from early to tardy (resp. tardy to early). From this we can also conclude that by interchanging the earliness and tardiness weights of a job, the value of the objective function remains unchanged (because ET_i is identically zero in an optimal solution).

Due to the above result, by interchanging the earliness and tardiness weights of all jobs i for which $w_i^+ > w_i^-$, we can obtain an optimal solution at the same cost as the original problem, but with no jobs tardy. Which is equivalent to a scheduling problem with only earliness in which the earliness weight for each job i is defined as Min $\{w_i^-, w_i^+\}$; i = 1,2,...,n.

Q.E.D.

Thus, the relaxed boundary conditions are constructed for each period k by considering a two period scheduling horizon, which is represented as follows:
For k = 1 (first period), the duration of the first period equals Δ_1, and the second period equals V, otherwise, the duration of the first period equals V, and the second period equals Δ_k. The weights are transformed by replacing the unit earliness and unit tardiness weights by a single weight w_i as follows (theorem 3):

$$w_i = \begin{cases} w_i^- & \text{if } d_i = 1 \text{ (Job due in first period)} \\ \text{Min}\{w_i^-, w_i^+\} & \text{otherwise} \end{cases} \tag{6.12}$$

The optimal cost for each period k is once again obtained using the Branch and Bound algorithm over this two-period horizon. Owing to the small size of this problem, it is usually computed quite fast. The Branch and Bound algorithm is also capable of considering a given set of jobs that are previously scheduled to complete in a period, while computing the optimal cost of the remaining jobs.

Although, in the scheduling problem for each k, one could consider a scheduling horizon of more than two elementary periods to obtain better estimates, this is avoided for the sake of efficiency. The computational efficiency of this bound is further improved by using a "quick bound procedure," detailed in section 6.3.1. The motivation for this procedure is that even though the problem for two elementary periods is a small problem, it is still NP-complete, and could be time consuming when the number of parts due in each period increases.

The above process is repeated for each of the periods (k=1,2,...z). The lower bound for all remaining jobs is the summation of the optimal costs for each of the periods. The closeness of the bound to the actual costs was found to be "good" in most cases, especially when overloaded (i.e., total duration of jobs due at the end of a period exceed Δ_k) and underloaded elementary periods bordered each other. However, when the load profile was triangular in nature (i.e., when most jobs were due in the initial (or ending) periods), the estimates obtained were fairly below the actual costs.

For the test problems considered in table 6.1, the quality of lower bound is assessed by: (i) the average percentage under-estimation of the initial lower from the optimum, and (ii) the proportion of times (as a percentage) the bound was within 10% below the optimum. These results are presented in table 6.2.

Problem Instances				Avg. % under-estimation	Proportion of times
n	$\overline{N}$	z	Runs	from Optimum	bound $\geq 0.9 \times$ Opt.
2	9.13	3	270	33%	73%
2	14.23	5	405	39%	68%
4	9.45	3	270	34%	45%
4	14.94	5	405	38%	40%
6	9.42	3	270	31%	47%
6	14.7	5	405	36%	42%

Table 6.2 : Evaluation of the quality of the initial lower bound

6.3.1 Quick bound procedure

Once again, we consider each elementary period separately. For each period k, we determine the set of unscheduled jobs with a due date at the end of period k. The difference here is that instead of determining the optimal cost of scheduling these jobs in period k with unconstrained boundary conditions, we estimate it by a lower bound, given the set of jobs that are previously scheduled completion in k.

Let $\mathcal{C}_k$ denote the set of jobs that are already scheduled to complete in period k of duration Δ_k, i.e. $\mathcal{C}_k = \{j \mid x_j = k$, and is scheduled$\}$. Let TC_k denote the total processing time of these jobs, and let L denote the longest job among them, i.e. $t_L > t_j$, $\forall j \in \mathcal{C}_k$, $j \neq L$. Let $\mathcal{R}_k$ denote the set of unscheduled jobs that are due at the end of period k, i.e. $\mathcal{R}_k = \{i \mid d_i = k$, and is unscheduled$\}$. Let TR_k denote the total processing time of these jobs, and let $\mathcal{L}_k$ denote the sub-set of jobs having processing time greater than job L, i.e. $\mathcal{L}_k = \{i \in \mathcal{R}_k$, and $t_i > t_L\}$. Now, $TC_k + TR_k$ represents the total processing time for the jobs belonging to $\mathcal{C}_k \cup \mathcal{R}_k$. If we assume that the completion time of job L is the earliest possible (just at the beginning of the period k), then the minimal processing time for jobs outside k equals $Z_{k,L}$, where $Z_{k,L} = (TC_k + TR_k) - (\Delta_k + t_L)$. If $Z_{k,L} \leq 0$, the quick bound for period k equals zero. Otherwise, we consider the jobs belonging to $\mathcal{R}_k$ ordered in nonincreasing (w_i/t_i) ratio to be processed in duration $Z_{k,L}$; partial processing of jobs is permitted in $Z_{k,L}$. Let $V_{k,L}$ be the cost of the jobs assigned to be processed in $Z_{k,L}$ (see algorithm 2 for details). If $TC_k > \Delta_k$, then $V_{k,L}$ is the lower bound, because $Z_{k,L}$ is the minimal processing time for jobs outside k. Otherwise, we can also attempt to schedule completion of job i, $i \in \mathcal{L}_k$, at the earliest time within the period k. Thus, the minimal processing time for jobs outside k in this case is $Z_{k,i}$, where $Z_{k,i} = (TC_k + TR_k) - (\Delta_k + t_i)$. If $Z_{k,i} \leq 0$, the quick bound for period k equals zero. Otherwise, we consider the jobs belonging to $\{\mathcal{R}_k \setminus i\}$ ordered in nonincreasing (w_i/t_i) ratio to be processed in duration $Z_{k,i}$; partial processing of jobs is permitted in $Z_{k,i}$. Let $V_{k,i}$ be the cost of the jobs assigned to be processed in $Z_{k,i}$. The lower bound for the period k is then given by $\text{Min}\{V_{k,i}, \forall i \in \mathcal{L}_k, V_{k,L}\}$.

The above process is repeated for each of the periods (k=1,2,...z). The lower bound for all remaining jobs is the summation of the "quick bounds" for each of the periods. We refer to this as the "quick bound procedure." This quick bound procedure can be employed for: (i) the computation of lower bound in the Branch and Bound algorithm for the entire scheduling problem, and/or (ii) the computation of the lower bound of section 6.3. Note that in case (i), we need to consider z periods, while in case (ii), we need consider only one period (observe the number of periods in the horizon of each case).

The quick bound is computed faster than the lower bound presented earlier, but there is a compromise on the quality as it further underestimates the lower bound. Although it is difficult to comment in greater detail about the quality of the bound, and the reduction in search space caused thereby, in table 6.3 we present the influence of the use of this bound on the number of nodes explored for a few arbitrary problem instances. Thus, considering these results, we employ the quick bound only in case (ii), i.e. in the Branch and Bound algorithm for each period in the lower bound procedure.

Problem instance	LB = 0	Quick bound procedure	Lower bound procedure
1	2,427	1,932	1,278
2	12,516	6,403	4,603
3	5,146	2,221	1,626
4	45,279	25,141	12,560

Table 6.3 : Influence of bound procedure on the number of nodes explored

The detailed algorithm of the quick bound procedure is presented in Algorithm 2.

Algorithm 2:

Notation :

Δ_k = duration of the k-th period

$\mathcal{C}_k = \{j \mid x_j = k$, and is scheduled$\}$

$\mathcal{R}_k = \{i \mid d_i = k$, and is unscheduled$\}$,and is an ordered set, ordered in nonincreasing (w_i/t_i)

L is the longest job belonging to $\mathcal{C}_k$, i.e. $t_L > t_j, \forall j \in \mathcal{C}_k, j \neq L$

$\mathcal{L}_k = \{i \in \mathcal{R}_k$ and $t_i > t_L\}$

QB_k is the quick bound for the k-th period.

```
For k = 1 to z do
        Determine $\mathcal{C}_k$ and $\mathcal{R}_k$;
        $TC_k = \sum_{j \mid j \in \mathcal{C}_k} t_j$;
        $L = \underset{j \mid j \in \mathcal{C}_k}{\text{Argmax}}\{t_j\}$; undefined if $\mathcal{C}_k = \varnothing$, with $t_L = 0$;
        $TR_k = \sum_{i \mid i \in \mathcal{R}_k} t_i$;
        Determine $\mathcal{L}_k$;
        $Z_{k,L} = (TC_k + TR_k) - (\Delta_k + t_L)$;
        if($Z_{k,L} \leq 0$)
                $QB_k = 0$; exit;
        else
                $V_{k,L}$ = schedule_completions($Z_{k,L}, \mathcal{R}_k$);
        if($TC_k > \Delta_k$ or $\mathcal{L}_k = \varnothing$)
                $QB_k = V_{k,L}$;
        For $i \mid i \in \mathcal{L}_k$ do
                $Z_{k,i} = (TC_k + TR_k) - (\Delta_k + t_i)$;
                if($Z_{k,i} \leq 0$)
                        $QB_k = 0$; exit;
                else
                        $V_{k,i}$ = schedule_completions($Z_{k,i}, \mathcal{R}_k \setminus i$);
        end;
        $QB_k = \text{Min}\{V_{k,i}, \forall i \in \mathcal{L}_k, V_{k,L}\}$;
end.
```

Procedure that returns the cost of processing an ordered set of jobs $\mathcal{R}$ in duration Z.

schedule_completions(Z, $\mathcal{R}$)

```
/*      Z is the duration in which jobs have to be processed        */
/*      $\mathcal{R}$ is an ordered set of jobs to be processed in Z; with $(w_1/t_1) \leq (w_2/t_2) \leq \bullet \leq (w_n/t_n)$  */
begin
        cost = 0; i = 0;
        while(Z > 0) do
                i = i + 1;
                if($t_i$ > Z)
/*                      cost = cost + $w_i$ ($Z/t_i$);      */
                        Z = 0;
                else
                        cost = cost + $w_i$;
                        Z = Z - $t_i$;
        continue;
        return(cost);
end.
```

7 Numerical Results

This section is devoted to performance evaluation of the branch and bound algorithm. At first, test examples relating to part data were constructed. For each example, a set of random demands was generated using different starting seeds and parameters. The details of the part examples and generation of this demand is presented in the following.

Table 7.1 presents the characteristics of the part related data for nine cases. For each case, 2, 4 and 6 part types (n) were generated, and thus, resulting in 27 examples. Cases 1.1 through 1.3 are the ones for which $w_i^- > w_i^+ \; \forall i$, and with varying processing times. Cases of 2 are the ones for which $w_i^- < w_i^+ \; \forall i$, and cases of 3 are the ones for which $w_i^- > w_i^+$ in general, but not consistently. While generating these data it is assumed that the length of the elementary periods, Δ, equals 1 unit of time. Thus, the processing times for parts are normalized in a way, and consistent with the assumption that $t_i < \Delta, \forall i$.

For each example, scheduling problems of varying degree of difficulty were constructed as follows. The parameters most relevant to the problem complexity are: (i) the number of elementary periods in the horizon, z, (ii) the number of elementary periods in which the total job processing time required is greater than Δ, denoted by zz, (iii) the percentage by which the total processing processing time required is greater than Δ in the type of periods described in (ii), denoted by f, and (iv) the average processing time required per elementary period, denoted by ff. The details of these parameters are presented in table 7.2. Finally, for each set of parameters and example, 5 random demand streams were generated.

Cases	n	Earliness Penalties	Tardiness Penalties	Processing time/part
1.1	2, 4, 6	U(1,10)	U(50,100)	U(.18,.38)
1.2	2, 4, 6	U(1,10)	U(50,100)	U(.25,.45)
1.3	2, 4, 6	U(1,10)	U(50,100)	U(.30,.45)
2.1	2, 4, 6	U(50,100)	U(1,10)	U(.18,.38)
2.2	2, 4, 6	U(50,100)	U(1,10)	U(.25,.45)
2.3	2, 4, 6	U(50,100)	U(1,10)	U(.30,.45)
3.1	2, 4, 6	U(20,60)	U(30,70)	U(.18,.38)
3.2	2, 4, 6	U(20,60)	U(30,70)	U(.25,.45)
3.3	2, 4, 6	U(20,60)	U(30,70)	U(.30,.45)

U(•,•) : Uniform Distribution.

Table 7.1 : Part related data for 10 examples

Cases	n	z	zz	f	ff
1	2, 4, 6	3	1,2	1.2, 1.4, 1.6	0.9
2	2, 4, 6	5	1,2,3	1.2, 1.4, 1.6	0.9

n: number of part types, z: number of periods, zz: number of overloaded periods, f: overload factor, and ff: horizon load.

Table 7.2 : Parameters selected in the scheduling problems

Thus, for each of the 2, 4 and 6 part type problem with 3 periods, a total of 270 problems were solved, and with 5 periods, a total of 405 problems were solved. Figure 7.1 presents the various complexities, on a logarithmic scale, as a function of the number of jobs in the scheduling problem. We recognize that the problem is NP-complete, thus the "brute-force complexity" is $O((z+1)^N)$, e.g., for a 3 period problem it is $O(4^N)$. This fact is indicated by the top-most curve in each graph. Due to the theorem 1, which imposes partial ordering on jobs of the same part type, the complexity of the problems is reduced; this is defined as "pruned complexity." These are indicated by the middle curve in each graph. The have the following observations to make: (i) the growth of pruned complexity w.r.t. N is slower than the brute-force complexity; e.g. for 2 part type 3 period problem the pruned complexity is $O(1.56^N)$ as opposed to the brute-force complexity that is $O(4^N)$, (ii) for the same horizon, the pruned complexity increases as the number of part types increases; this is intuitive as the number of jobs between which precedence relations exist decreases, and (iii) as the number of periods in the horizon (z) increases for a fixed number of part types (n), the growth rate of pruned complexity w.r.t. N may decrease or increase; the decrease is observed due to the higher number of job precedences, and the increase is due to the increase in z, thus the net effect is not conclusive. Finally, the bottom curve in each graph indicates the complexity of the branch-and-bound algorithm. It is the number of nodes explored by the algorithm in order to reach the optimum, on a logarithmic scale, as a function of N. Although the growth of nodes explored is still exponential, the rate of growth is much less than either the brute-force or pruned complexity. For instance, in the 6 part type 3 period problem, the brute-force complexity is $O(4^N)$, the pruned complexity is $O(2.2^N)$ and the algorithmic complexity is $O(1.6^N)$. The algorithmic complexity observed for the range of examples considered is between $O(1.6^N)$ and $O(1.3^N)$.

The number of nodes in the list remained less than 60 in the examples tried, which indicates that the memory requirements are reasonable. Of course, in general, the number of nodes in the list increases as the size of the search space increases. Figure 7.2 is an attempt to correlate the number of nodes in the list to the logarithm of the number of nodes explored. Although there is a fair scatter, one can appreciate that the number of nodes in the list grows linearly with respect to the logarithm of the number of nodes explored.

The other parameters of interest that influence the number of nodes expanded by the algorithm are the overload factor (f) and the average load in the horizon (ff). The statistics on these were rather inconclusive; graphs between f (resp ff) and the logarithm of the number of nodes expanded had a fair degree of scatter. However, we observed that in general, the complexity of the algorithm increases with respect to these parameters. The c.p.u. time for the branch and bound algorithm, implemented in C on the SUN/SPARC station, was generally proportional to the number of nodes explored. A limit of 600 seconds was imposed on the execution, in which case, the best solution was reported. Only two problems, one for a 2 part type 5 period problem and the other for a 6 part type 5 period problem remained unsolved out of the 2,025 problems attempted.

7.1 MILP Solutions

In this section, we present some numerical results obtained for the MILP formulation presented in section 5. The MILP was solved by the OSL (Optimization Subroutine Library) on the IBM 3081 mainframe. The MILP solver of OSL also employs a basic version of the branch and bound method; default values of all control variables were chosen. The data was provided in the MPS (Mathematical Programming System) format, compatible with MPSX/370. The number of problems solved was relatively low due to the high computation costs. The purpose of this exercise is simply to demonstrate the concept, and not to perform a statistical comparison of the MILP formulation to the branch and bound algorithm.

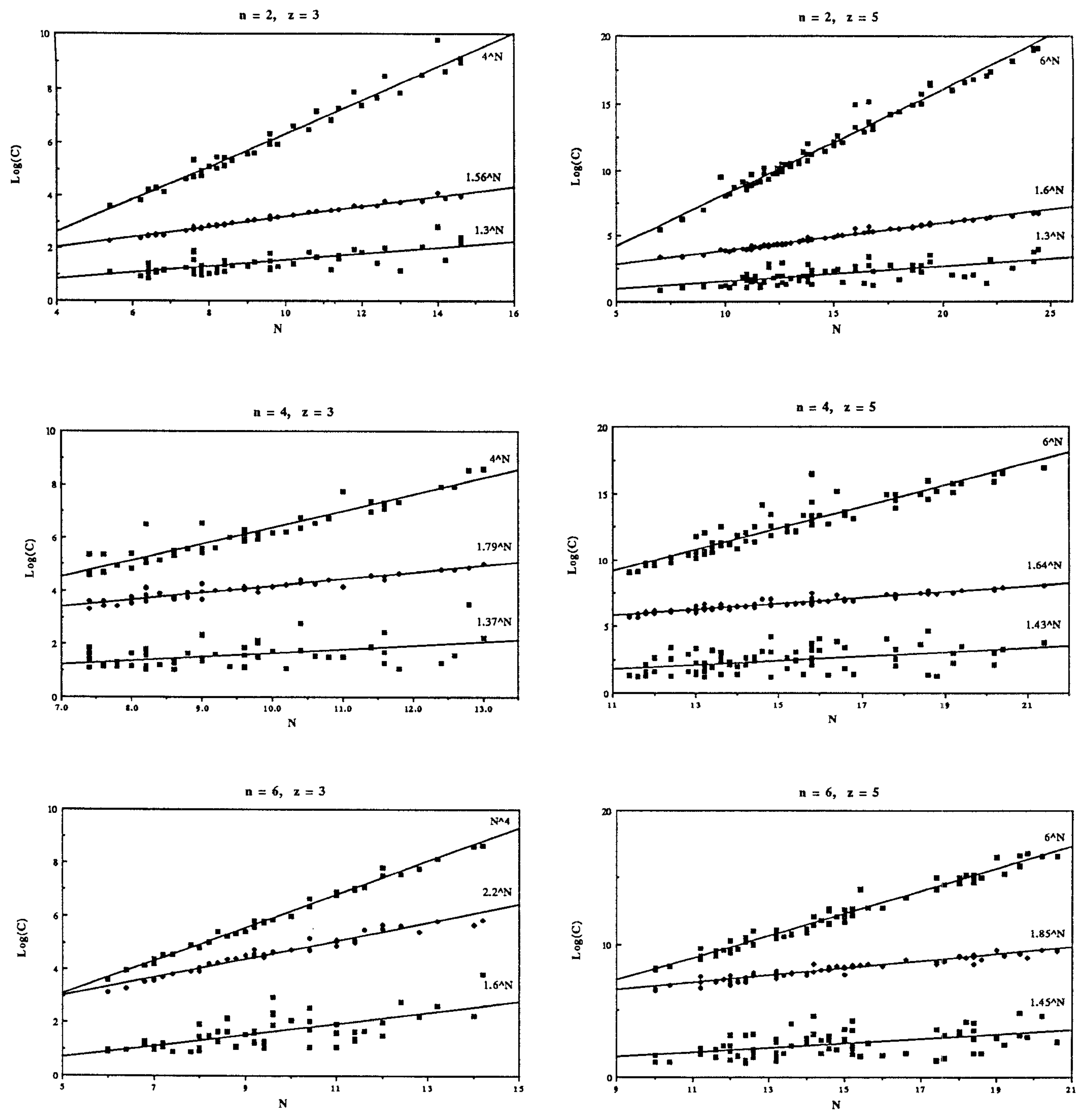

Figure 7.1 : Brute-force, pruned and algorithmic complexities versus N

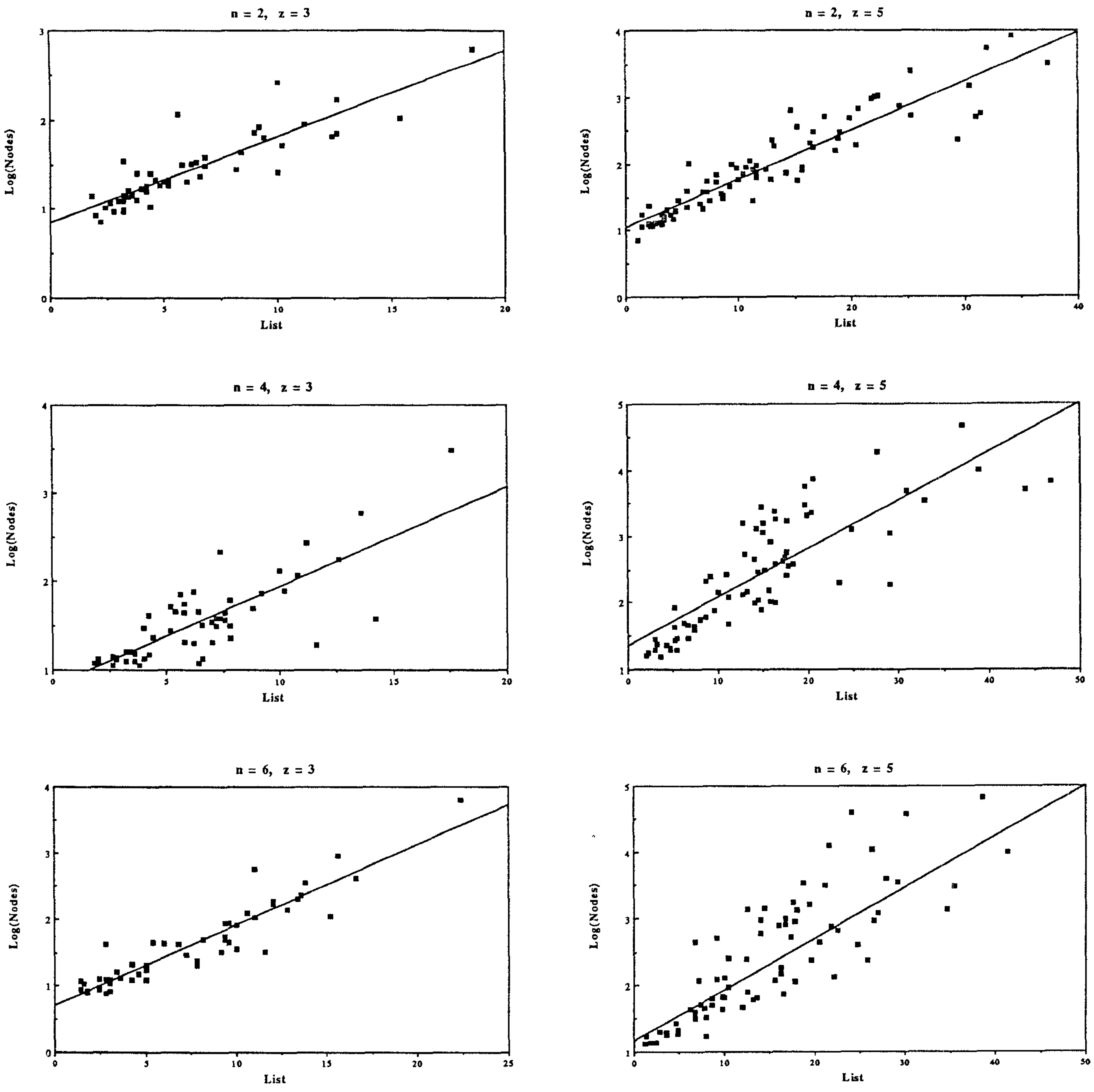

Figure 7.2 : Relation between number of nodes in the List and number of nodes explored

Problem Instance							MILP Solution			Branch and bound	
No.	N	n	z	zz	f	ff	Nodes	Iterations	sec.	Nodes	sec.
1	22	2	5	1	1.4	0.9	5172	14255	358	182	1.68
2	24	2	5	3	1.4	0.9	24919	81921	1903	163	1.3
3	18	4	5	1	1.4	0.9	1800	5610	116	164	0.75

Table 7.3 : MILP solutions of 3 problem instances

Table 7.3 presents the MILP solutions for 3 arbitrary problem instances. The MILP result is characterized by the number of nodes explored by the OSL routines, the number of iterations, and the c.p.u. time in seconds for the IBM 3081. The branch and bound algorithm is characterized by the number of nodes explored, and the c.p.u. time for the implementation discussed previously on SUN/SPARC station.

9 Conclusions

We consider a one-machine scheduling problem with discrete, weighted earliness and tardiness. The problem considered is somewhat new, and finds place is some real-world applications. We prove that this problem is NP-complete, and present results for partial job ordering. A Mixed Integer Linear Programming (MILP) formulation of this problem is also developed. The major contribution of this work is the development of branch-and-bound scheme that solves this problem optimally. Heuristic rules that provide an initial feasible solution provide the upper bound to the scheme. The performance evaluation of these rules is also studied, which suggests their applicability to solving the scheduling problem heuristically. Two lower bound schemes have been developed and are based on a "boundary-relaxation" approach. Numerical observations indicate that these lower bound help reduce the number of nodes explored in the search procedure. The branch and bound scheme can reach the optimum in problems of reasonable complexity. Numerical results indicate that the growth of complexity w.r.t. the number of jobs to be scheduled is much slower than the original problem. For the problems attempted, that were $O(4^N)$ and $O(6^N)$ in complexity, the branch and bound algorithm explored nodes between $O(1.6^N)$ and $O(1.3^N)$. Thus, we can hope to solve problems of larger dimension with the help of this scheme. Furthermore, since the search is based on a depth-first strategy, for "hard" problems the search can be terminated prior to successful termination to provide a "good" solution for practical applications. We hope that some of these ideas can be extended to general job-shop scheduling with discrete earliness and tardiness.

Acknowledgements

The authors acknowledge the Computer Science Center at the University of Maryland for the access of the IBM 3081 mainframe.

References

[1] Abdul-Razaq, T.S. and Potts, C.N. (1988) : "Dynamic Programming State-Space Relaxation for Single-Machine Scheduling," Journal of Opl. Res. Soc., Vol. 39, No. 2, pp 141-152, 1988.

[2] Baker, K.R. (1974) : "Introduction to Sequencing and Scheduling," Wiley, 1974.

[3] Baker, K.R. and Scudder, G.R. (1990) : "Sequencing with Earliness and Tardiness Penalties : A Review," Operations Research, Vol. 38, No. 1, pp. 22-36, 1990.

[4] Chand, S. and Schneeberger, H (1988) : "Single Machine Scheduling to Minimize Weighted Earliness Subject to No Tardy Jobs," Int. J. of Opns. Res., Vol 34, pp. 221-230.

[5] Conway, R.W., Maxwell, W.C. and Miller, L.W. (1967) : "Theory of Scheduling," Addison-Wesley, 1967.

[6] Emmons, H. (1969) : "One-Machine Scheduling to Minimize certain Functions of Job Tardiness," Operations Research, Vol. 17, pp 701-715, 1969.

[7] Ferris, M.C. and Vlach, M. (1992) : "Scheduling with Earliness and Tardiness Penalties," Nav. Res. Logist., Vol 39, pp. 229-245, 1992.

[8] French, S. (1982) : "Sequencing and Scheduling," Ellis Horwood Limited (1982).

[9] Fry, T.D., Armstrong, R.D. and Blackstone, J.H. (1987) : "Minimizing Weighted Absolute Deviation in Single Machine Scheduling," IIE Transactions, Vol. 10, No. 4, pp 445-450, 1987.

[10] Garey, M.R. and Johnson, D.S. (1979) : "Computers and Intractability : A Guide to the Theory of NP-Completeness," W.H. Freeman and Co., New York, 1979.

[11] Garey, M.R., Tarjan, R.E. and Wilfong, G.T. (1988) : "One-Processor Scheduling with Symmetric Earliness and Tardiness Penalties," Math. Opns. Res., Vol 13, pp. 330-348.

[12] Gupta, S.K. and Kyparisis, J. (1987) : "Single Machine Scheduling Research," OMEGA Int. J. Mgmt. Sci., Vol. 15, No. 3, pp. 207-227, 1987.

[13] Hall, N.G. and Posner, M.E. (1991) : "Earliness-Tardiness Scheduling Problems, I: Weighted Deviation of Completion Times about a Common Due Date," Operations Research, Vol. 39, No. 5, pp 836-846, 1991.

[14] Hartigan, J.A. (1957) : "Clustering Algorithms," John Wiley & Sons, 1975.

[15] Lakshminarayan, S., Lakshman, R., Papineau, R.L. and Rochette, R. (1978) : "Optimal Single-Machine Scheduling with Earliness and Tardiness Penalties," Opns. Res., Vol 26, pp. 1079-1082, 1978.

[16] Lawler, E.L. (1977) : "A "Pseudopolynomial" Algorithm for Sequencing Jobs to Minimize Total Tardiness," Ann. Discrete Math., Vol 1, pp. 331-342, 1977.

[17] Ow, P.-S. and Morton, T.E. (1989) : "The Single Machine Early/Tardy Problem," Management Science, Vol. 35, No. 2, pp 177-191, 1989.

[18] Sen, T. and Gupta, S.K. (1984) : "A State-of-the-Art Survey of Static Scheduling Research Involving Due Dates," OMEGA, Vol 12, No. 1, pp. 63-76, 1984.

[19] Sidney, J. (1977) : "Optimal Single-Machine Scheduling With Earliness and Tardiness Penalties," Opns. Res., Vol 25, pp. 62-69, 1977.

[20] Smith, W.E. (1956) : "Various Optimizers for Single-Stage Production," Nav. Res. Logist. Quart., Vol 3, pp. 59-66, 1956.