TECHNICAL RESEARCH REPORT

SYSTEMS RESEARCH CENTER

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

Performance Evaluation of a Hierarchical Production Scheduling Policy for a Single Machine with Earliness and Tardiness

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

TR 90-72

Performance Evaluation of a Hierarchical Production Scheduling Policy for a Single Machine with Earliness and Tardiness

Rakesh Nagi [1], George Harhalakis [1], Jean-Marie Proth [2]

[1]Department of Mechanical Engineering & Systems Research Center, University of Maryland, College Park, MD 20742, U.S.A.
[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 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 penalty functions that are staircase-type. No preemption of jobs is permitted, and idle time may be inserted. This problem is NP-complete. A branch-and-bound scheme that solves the above mentioned problem optimally is presented. We then propose a hierarchical scheduling policy under the assumption that tardiness costs are much greater than earliness costs for all jobs. We also assume that the system is able to meet the production requirements on the average and the scheduling horizon is long enough to absorb cyclicity. The hierarchy is composed of two levels : (i) the high level, and (ii) the low level. The concept of rolling horizon is employed at the high level, which solves the flow control under constraints of no tardiness. The low level, prioritizes jobs resulting from the solution of the high level over a short term horizon. Numerical results relating to the comparison of the performance of this hierarchical policy with the branch-and-bound scheme are presented.

Key words : Production/Scheduling, Hierarchical models, 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 literature related to this problem. Previously, the research focus was on single *(regular)* performance measures, that are nondecreasing in job completion time. 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.

We consider a specific problem of scheduling N jobs to minimize the total earliness and tardiness penalties on a single stage manufacturing system. The problem treated here differs from the previous work in this area in that the earliness and tardiness penalties are discrete as opposed to being continuous. We consider the scheduling horizon (over which the N jobs have to be sequenced) to be discretized into elementary periods. For instance, the scheduling horizon is a week, then the elementary period could be a day. Jobs can have due dates at the end of an elementary period. Completion of a job in a period prior to that of its due date entails it to be held in inventory till delivery, while failure to complete a job by its due date causes backlogging with delivery possible only at the end of the period in which it is completed. As the basic idea of earliness penalty stems from discouraging the holding of finished parts in inventory, we believe in penalizing the job by the number of periods (an integer) it spends in the inventory, weighted appropriately by its holding cost per period. Analogously, tardiness penalty discourages violation of due dates, we believe in penalizing the job by the number of periods (an integer) by which it is delayed, weighted appropriately by its backlogging cost per period. We do not penalize jobs completing within the periods at the end of which it is due as they can be held on the shop floor or transported to the shipping area without having to be checked into the inventory. Furthermore, if inventory book-keeping and deliveries are performed at the end of the day, our assumptions of discrete penalization find good ground. Thus, we employ staircase penalty functions for earliness and tardiness of each job opposed to linear or quadratic considered in the literature.

Thus, to summarize, we consider the problem of scheduling a set of initially available jobs, to be produced within a given scheduling horizon on a single machine to minimize the total earliness and tardiness penalties of all the jobs, subject to no preemption constraint. The scheduling horizon is discretized into elementary periods; the jobs have specified distinct due dates, and can be delivered at the end of a period. The penalty functions are discrete and weighted (staircase).

Owing to the nonregular objective function (i.e., that can decrease with the increase in completion times), important theorems (Emmon 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 develop a Branch-and-Bound scheme to obtain optimal results for the above stated problem. We will then demonstrate on this problem, the design and operation of a hierarchical production scheduling policy and its main advantages.

In the next section, we survey some of the related literature. In section 2, the problem formulation is presented, detailing first the manufacturing system, the demand, and the criteria of interest. The informal proof of NP-completeness is presented in section 3. In section 4, we describe the Branch-and-Bound scheme that solves the problem optimally, followed by the hierarchical model in section 5. The problem formulations at each level of the hierarchy are presented in section 6, and the algorithms for them detailed in section 7. Section 8 is devoted to the comparison between the Hierarchical approach and the Optimal algorithm.

1.1 Literature Review

Majority 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 Lakshimarayan *et al* (1978) present algorithms for minimizing the maximum penalty of early or late jobs. A comprehensive survey relating to earliness and tardiness (E/T) models can be found in Baker and Scudder (1990). They site a host of literature with different assumptions like : common due-dates, unweighted or

equal E/T weights, no insertion of idle time, etc. We restrict the references to the distinct due-date weighted total E/T model. Garey *et al* (1988) prove that this 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 technique 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 into determining a good sequence, followed by the optimal insertion of idle time. We are not aware of any literature related to periodic distinct due-date weighted total E/T problem with staircase penalty functions.

2 Problem Formulation

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

We consider a single-stage manufacturing system consisting of a single machine M capable of producing a set of n part types represented by $P = \{p_1, p_2, \ldots, p_n\}$. Let τ_j denote the processing time required on M to produce one unit of part type p_j; $j=1,2,\ldots,n$. We also assume data regarding average part production volumes is available, either historically or by forecast. Let n_j represent the average number of products of type p_j produced per unit time; $j = 1,2,\ldots,n$.

2.1 Demand

We assume that orders are accepted in a particular planning horizon. Let this horizon be denoted by H, which is composed of z elementary periods. An elementary period is any convenient period, like a day; this will be qualified later. A set of production orders are assumed to appear in every elementary period. Each order consists of the following information :

- Part type, i.e., an index $j \in \{1,2,\ldots,n\}$.
- Quantity, q_j, $q_j \in \mathbb{N}$.
- Due date, s_j, i.e., $s_j \in \{1,2,\ldots,z\}$.

An elementary period has a duration of Δ time units. It is assumed that $\tau_j << \Delta$. Furthermore, for simplicity, we assume that the delivery of orders take place at the end of an elementary period, and not during it.

Thus, for each elementary period, we know the part types and their respective quantities to deliver. Let d(j,k) denote the number of units of part type p_j to be delivered at the end of the k-th period.

2.2 Criteria

We consider the total weighted earliness and tardiness penalties in the scheduling model. Finished Parts Inventory and Backlogging costs are the criteria to be minimized. The first criterion penalizes the system for overproducing parts or holding finished parts in the inventory before delivery, while the second one penalizes the system for not being able to meet orders at the desired due-dates (tardiness cost). These criteria together result in a nonregular performance measure.

Let c_j^+ represent the cost associated with holding in stock one unit of part type p_j for one elementary period. Equivalently, let c_j^- represent the cost associated when one unit of part type p_j is delayed by one elementary period. In our approach, we are not restricted by assumptions of the kind : *"agreeable"* weighting of jobs (Lawler 1977). In other words, backlogging costs, inventory costs, and processing times need not hold any specific relationship.

We now present the problem formulation as follows. Let there be z elementary periods in the scheduling horizon. 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 in terms of jobs as opposed to the one in parts introduced earlier. Let the total number of jobs to be scheduled be N, (Note : $N = \Sigma_k \Sigma_j\ d(j,k)$). Job i is described by : (1) t_i, denoting the processing time, and (2) d_i, denoting the due date ($d_i \in \{1,2,\ldots,z\}$). Let C_i denote the completion time of job i. Then, x_i, denoting the elementary period in which job i is completed can be computed by $\lceil C_i/\Delta \rceil$, where $\lceil \bullet \rceil$ denotes the smallest integer greater than or equal to $\bullet$. Let E_i and T_i represent the earliness and tardiness, respectively, of job i; these are detailed as : $E_i = \mathrm{Max}(d_i - x_i, 0) = (d_i - x_i)^+$ and $T_i = \mathrm{Max}(x_i - d_i, 0) = (x_i - d_i)^+$, respectively. Note : $E_i, T_i \in \mathbb{N}$. We associate with each job, i, and earliness penalty w_i^+, and a tardiness penalty w_i^- when i is completed in a period before (d_i-1) and after (d_i+1), respectively, its due date d_i. Let S represent a schedule.

The scheduling problem can formally be stated as follows :

Minimize : $$\text{Cost}(S) = \sum_{i=1}^{N} (w_i^+ \times E_i + w_i^- \times T_i) \tag{2.1}$$

Subject to : $$[C_r - t_r, C_r) \cap [C_s - t_s, C_s) = \emptyset \; ; \quad \forall r, \forall s, \text{ and } r \neq s. \tag{2.2}$$

$$C_i - t_i \geq 0 \; ; \quad \forall i. \tag{2.3}$$

$$C_i \leq z \times \Delta ; \quad \forall i. \tag{2.4}$$

3 NP-completeness

In this section, we informally demonstrate that the scheduling problem detailed in section 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 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 = 2.
(2) Let duration of first period, $\Delta = B$, letting duration for second period = ∞.
(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 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 D is also NP-complete.

q.e.d.

4 The Branch and Bound Algorithm

Owing to the discrete nature of penalty functions, nonregular criterion as well as the introduction of idle time, it is difficult to come-up with job ordering theorems. At this time, we are unaware of any applicable job ordering theorems for this problem, so the present version of the Branch and Bound algorithm does not employ any pruning devices. Although for the jobs that correspond to the same part type we can introduce precedence relationships, this was not employed as the algorithm provided one among the multiple optima in a reasonable time for most cases.

We employ four heuristic rules to obtain feasible solutions of the scheduling problem; the lowest cost serving as the initial upper bound. These heuristic rules, and their performances are detailed in section 4.1.

Jobs are ordered according to earliest due dates, we break ties with lower part number first. The algorithm begins with the scheduling of the last job (i.e., the one with the latest due date), and precedes in a backward fashion until the first job is scheduled. At any step of the algorithm, we expand the deepest node in the tree (a partial schedule with the maximum number of scheduled jobs), ties broken by lowest value first, provided its value is strictly less than the current upper bound. The value of a node is defined as the cost of the partial schedule so far, plus a lower bound estimate of the cost of scheduling the remaining jobs. We detail the calculation of the lower bound in section 4.2. For the node being expanded, we have a set of tentatively scheduled jobs (i.e., x_i are known), and we explore the possibility of scheduling the completion of the subsequent part in each of the periods. For each admissible assignment, we can then compute the value of the new node, and add it to the list of unmarked nodes provided its value is strictly less than the current upper bound, The parent node is then marked. The upper bound is updated when a node corresponding to the first job is marked. The algorithm stops when there is no unmarked node with a value strictly less than the current upper bound. The current upper bound is now the optimal cost, and the corresponding schedule is the optimal schedule. Note that the depth first strategy is preferred over the best first one in order to update the upper bound faster, consequently, helping in restricting the search.

4.1 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 most appropriate available 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 $\mathrm{Max}(w_i^-, w_i^+)/t_i$.
Rule 4 : Jobs are arranged in nonincreasing order of $\mathrm{Max}(w_i^-, w_i^+)$.

At each step, the heuristic selects the first unscheduled job and schedules it in the most appropriate location. The most appropriate location is defined as the one that contributes least to the cost function. In other words, if feasible, schedule completion of a job in the period corresponding to its due date, else try scheduling completion in the period before (after) its due date when earliness penalty is less than (greater than) tardiness penalty. If the job remains unscheduled, schedule completion in a first feasible period before or after its due date that imposes least penalty.

Several trials were performed with varied number of jobs, penalties, processing times and due-dates. In 50% of the cases, one of the heuristics was able to provide a bound within 10% of the optimal. In table 4.1, we present the number of times (in percentage) when a rule provided the lowest cost. Note that the summation is greater than 100 because for some cases more than one rule provided the lowest cost.

Rule 1	Rule 2	Rule 3	Rule 4
75%	43%	56%	12%

Table 4.1 : Number of times (percentage) bound found by a rule

4.2 Lower bound Estimate of scheduling remaining jobs

We consider an elementary period k, and determine the optimal cost to schedule the remaining jobs due at the end of this period $\{i \mid d_i \leq k$, and is unscheduled$\}$ with unconstrained boundary conditions from the neighboring period(s). Boundary conditions of the scheduling horizon (external) are still respected by the first elementary period. The optimal cost for one period is once again obtained using the Branch and Bound algorithm. Owing to the small size of this problem, it is usually

computed quite fast. Furthermore, the unconstrained version can easily be converted into a problem having a single criterion by replacing the unit earliness and unit tardiness weights by a single weight w_i as follows :

$$w_i = \begin{cases} w_i^- & \text{if } d_i = 1 \text{ (Job due in first period)} \\ \text{Min}\{w_i^-,\ w_i^+\} & \text{otherwise} \end{cases} \tag{4.1}$$

The computational efficiency of this bound is further improved by using a "quick bound procedure," detailed in section 4.2.1.

The above process is repeated for each of the periods (k=1,2,...z). The lower bound estimate for the entire remaining jobs is the summation of the optimal costs for each of the periods. Lower bounds computed in this manner were usually obtained quickly. 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) and underloaded elementary periods bordered each other. However, when the load profile was triangular in nature, the estimates obtained were fairly below the actual costs.

4.2.1 Quick bound procedure

Once again, we consider an elementary period k, and try to schedule the remaining jobs due at the end of this period $\{i \mid d_i \leq k$, and is unscheduled$\}$ with unconstrained boundary conditions from the neighboring period(s); the difference here is that instead of determining the optimal scheduling cost, we estimate it by a lower bound that permits partial job penalization. In other words, if a job begins processing in period k, but, is completed in k+1, then, we prorate the penalization by the duration the job spend in period k+1. This lower bound can be computed efficiently by considering jobs ordered according to nonincreasing (w_i/t_i) ratio to be included in the period. The above process is repeated for each of the periods in the horizon under consideration. The lower bound estimate for the entire 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, and/or (ii) the lower bound of section 4.2. Note : in case (i) the horizon under consideration will have z periods, while in case (ii), it will have 2 periods.

Although it is difficult to comment in greater detail about the quality of the bound, and the reduction in search space caused thereof, we present in table 4.2, the influence of

the use of this bound on the c.p.u. times required by random problem instances (see Table 8.1 and 8.2 for instance details).

Problem instance (see table 8.1 & 8.2)	LB = 0	Quick bound procedure	Lower bound procedure
1(a)	10.38 sec	5.75 sec	3.73 sec
8(a)	30.41 sec	21.15 sec	6.58 sec
7(a)	unsolved	2166.58 sec	295.80 sec

Table 4.2 : Influence of bound procedure on 3 problem instances

5 Description of the Hierarchical Model

The hierarchical approach is a heuristic that is intended to overcome the intractability associated with the problem described in section 2. The original problem is decomposed into a set of simpler sub-problems that are solved sequentially in order to obtain the solution of the overall problem. Such models can provide results very fast. This benefit is however not obtained at no cost; the compromise is the quality of the solution.

Our objective is to develop a general hierarchical model that is not only capable of addressing the current problem, but the following issues associated with planning/scheduling problems as well :

(1) Unreliable machine : Machines are prone to failures or breakdowns. While the machine is under repair, it cannot process a part, and this leads to a reduction in capacity. We assume that the working period and repair periods are exponentially distributed random variables with means of MTBF and MTTR respectively; with MTBF and MTTR$<< \Delta$. Assumptions of nonpreemption still hold.

(2) Random demand : Although scheduling horizons are usually small with demand known deterministically, random events like order cancellations, revisions, expeditions, etc. are not unusual. Especially in a forecasted environment, the demand is subject to change due to uncertainty. In such cases, frequent recomputations of the already time consuming scheduling algorithm may not be possible. The hierarchical model can accomodate the uncertainty to a certain degree by aggregating similar parts.

In this section, we develop a hierarchical model under a set of assumptions that is applicable to a particular class of problems (to be detailed later). We will then

specialize the general hierarchical model to address the deterministic scheduling problem at hand and evaluate the perfromance of this model, i.e. compare it with the Branch and Bound scheme.

This hierarchical model intended to address the following class of problems. For these problems we assume that (i) $c_i^- >> c_i^+ \ \forall i$, or the backlogging costs are much greater than inventory costs in general, and (ii) in the horizon H, we are able to meet production requirements and absorb cyclicity (seasonality) of the demand. Assumption (i) is in fact consistent with the trend followed in industry, where backorders are penalized severely, and (ii) can be justified by a choice H such that for most typical demand pattern backward smoothing is possible [1].

The hierarchical model proposed consists of two levels : (i) High level, and (ii) Low level. The high level model is more aggregate, and the entities of relevance are part families. The low level model is detailed, and the entities of relevance are parts. Part families are constructed by grouping parts having similar processing times as well as similar holding costs. The part types are aggregated into part families by K-mean cluster analysis [] []. The advantages of aggregation in this manner allow for some demand uncertainty to be absorbed. The aggregate part family can be substituted by any part belonging to this family, this reduces the variance while retaining samilar processing durations and holding costs in the schedule. The set of f part families is represented by $F = \{f_1, f_2, \ldots, f_f\}$. Let θ_j denote the processing time required on M to produce one unit of family type f_j; j=1,2,...,f. Let v_j represent the cost associated with holding in stock one unit of family type f_j for one elementary period.

Now, for each elementary period, we can aggregate the demand of the parts to generate the demand of families. Let q(j,k) denote the number of units of family type f_j to be delivered at the end of the k-th period.

6 Problem Formulation

In this section, we describe the problems at the two levels of the hierarchy, and formulate them as optimization problems.

6.1 High Level Optimization Model

[1]See section 4.1.

At the high level of the hierarchy, we are interested in planning for the flow of part families. Given an aggregate production requirement plan for the planning horizon H, the objective is to determine a production flow plan for part families such that the system capacity as well as backlogging constraints are respected and the total holding cost is minimized. Note that we treat backlogging cost as a constraint here because of the two assumptions made in section 5, and the following reasons. It is assumed that in steady-state (infinite horizon), we are able to meet production requirements, i.e., the production requirement is less than or equal to the average system capacity. We propose that a rolling horizon is capable of emulating steady-state behavior, and the horizon chosen is large enough to absorb cyclicity (seasonality) of the demand. Thus, under these propositions, the backlogging constraint is not over-restricting at this level. This process is also called backward smoothing.

Let u(j,k), u(j,k) $\in$ IR^+, denote the number of units of family type f_j to be produced during the k-th period, and u(j,0) denotes the initial stock level of family type f_j. We define x(i,k) to be the excess production for part type p_i at the end of the k-th period, and is detailed in (6.1).

$$x(j,k) = \left[\sum_{t=0}^{k} u(j,t) - \sum_{t=1}^{k} q(j,t)\right] \tag{6.1}$$

The high level optimization problem can then formally be stated as follows :

Minimize : $$\sum_{j=1}^{f} v_j \times \sum_{k=1}^{z} x(j,k) \tag{6.2}$$

Subject to : $$\sum_{j=1}^{f} \theta_j \times u(j,k) \le \frac{MTBF}{MTBF+MTTR} \times \Delta; \qquad k = 1,2,\ldots,z \tag{6.3}$$

$$\sum_{t=0}^{k} u(j,t) \ge \sum_{t=1}^{k} q(j,t); \qquad j = 1,2,\ldots,m;\ k = 1,2,\ldots,z \tag{6.4}$$

By substituting for x(j,k) from equation 6.1 in objective 6.2, and realizing that the latter summation is a constant, the objective 6.2 can be reformulated as in equation 6.5.

Minimize : $$\sum_{j=1}^{f} v_j \times \sum_{k=1}^{z} \sum_{t=0}^{k} u(j,k) \tag{6.5}$$

If we replace $v_j \times$ u(j,k) by u'(j,k), the above linear programming problem can be transformed into the following :

Minimize : $$\sum_{j=1}^{f} v'_j \times \sum_{k=1}^{z} \sum_{t=0}^{k} u'(j,k) \tag{6.6}$$

Subject to : $$\sum_{j=1}^{f} u'(j,k) \le \frac{MTBF}{MTBF+MTTR} \times \Delta; \qquad k = 1,2,\ldots,z \tag{6.7}$$

$$\sum_{t=0}^{k} u'(j,t) \ge \sum_{t=1}^{k} q'(j,t); \qquad j = 1,2,\ldots,f;\ k = 1,2,\ldots,z \tag{6.8}$$

where : $q'(j,k) = v_j \times q(j,k)$, and $v'_j = v_j/\theta_j$.

Using the similar arguments as in Libosvar 1988, one can prove these problems to be equivalent. Interestingly, this problem can be optimally solved without having to formally solve the linear programming problem. In section 7, we propose the algorithm that optimally solves (6.6) subject to (6.7) and (6.8). Before we use this algorithm, we transformation of the demand, q(j,k), into an effective demand by netting against on hand stock, u(j,0).

6.2 Low Level Optimization Model

At the low level of the hierarchy, we are interested in scheduling parts on the machine under the real-time capacity constraints. As it was mentioned in section 5, the machine is unreliable and is subject to failure. When the machine is in repair state, it cannot produce parts, therefore the is no available capacity. However, when the machine is in working state, it can be utilized to full capacity. The problem at the low level is then to determine the sequence of parts to load in the low level horizon such that the immediate orders are satisfied with minimal backlogging, and the high level flow plan is respected as closely as possible. The low level horizon is composed of w elementary periods [1] . Note that it may not always be possible to satisfy the orders (with no backlogging) because of machine failures and the aggregate nature of the high level flow plan. The high level flow plan could turn out to be infeasible for a particular elementary period at the low level because of its aggregate nature. In fact, it depends on : (i) the closeness of the parts belonging to the same family, and (ii) the ratio of parts within families. We hope that "good" part families (aggregation) can ensure consistency of the high level plan. Generally, we can satisfy the short term production requirements on the average.

---

[1] If the low level horizon is composed of w elementary periods, and the high level horizon is composed of h low level horizons, we have : $w \times h = z$.

Let y(i,k), y(i,k) ∈ IN, denote the number of units of part type p_i produced during the k-th period, and y(i,0) denote the initial stock level of part type p_i. We define s(i,k) to be the deficit production for part type p_i at the end of the k-th period, and is detailed in (6.9).

$$s(i,k) = \mathrm{Max}\left\{\sum_{t=1}^{k} d(i,t) - \sum_{t=0}^{k} y(i,t), 0\right\} \tag{6.9}$$

We also define the occupation of machine M by a part type p_i by a binary variable $e_i(t)$:

$$e_i(t) = \begin{cases} 1 & \text{if part type } p_i \text{ is being operated by machine M at time t} \\ 0 & \text{otherwise} \end{cases} \tag{6.10}$$

The low level problem can then formally be stated as follows.

$$\text{Minimize :} \quad \sum_{i=1}^{n} c_i^- \sum_{t=1}^{w} s(i,t) \tag{6.11}$$

$$\text{Subject to :} \quad \sum_{i=1}^{n} e_i(t) \le \begin{cases} 1 & \text{if machine M is operational} \\ 0 & \text{otherwise} \end{cases} \qquad t \in R^+ \tag{6.12}$$

$$\sum_{i \mid p_i \in f_j} y(i,k) = u(j,k); \qquad j = 1,2,\ldots,f;\ k = 1,2,\ldots,z \tag{6.13}$$

Constraint (6.12) implies that a maximum of one part can occupy the machine when it is operational, while none can when it is under repair. Constraint (6.13) is the high level flow plan for part families. Note that the variables y(i,k) are integer while flow variables u(j,k) are continuous. Thus, we can schedule the beginning time of $p_i \in f_j$ such that only a portion of it is produced in the relevant period while it completes processing in the subsequent period. That is to say, a fractional part requirement leads to the scheduling of the part such that it is completed in the subsequent period. This however requires that there be no more than two partial u(j,k) for ever period k; k = 2,...,z-1 (one corresponding to the start and the other corresponding to end of the period). In this way we can respect the upper level constraints.

The problem detailed in this section, can also be solved using a heuristic rule which yields acceptable results, thus, we do not formally solve the linear programming type problem. In section 7, we propose this heuristic algorithm.

7 Algorithms

7.1 High level Algorithm

In this section, we detail the backward smoothing algorithm that solves (6.6) subject to (6.7) and (6.8).

If $F = \{1,2,\ldots,f\}$, we define an ordering function O, $O : F \rightarrow F$, with $v'_{o(i)} \geq v'_{o(j)}$ $\forall i < j$, $i,j \in F$.

```
for k = z down to 1 do
    Initialize :
    u(j,k) = 0 for j = 1,2,...,f;
    Available_capacity := MTBF/(MTBF+MTTR) × Δ;
    for i = 1 to f do
        j = O(i);
        load = θ_j × q(j,k);
        if(load ≤ Available_capacity) then
            Available_capacity = Available_capacity - load;
            u(j,k) = q(j,k)
        else
            u(j,k) = (Available_capacity/load) × q(j,k);
            if(k = 1) then
                "Infeasible Plan"; exit;
            q(j,k-1) = q(j,k-1) + (q(j,k) - u(j,k))
            for s = i+1 to f do
                j = O(s);
                q(j,k-1) = q(j,k-1) + q(j,k);
```

Although this algorithm is optimal, it could result in more than two partial jobs in an intermediate period, thus making it difficult for the low level to respect the constraint (6.13). Therefore, we modify the above algorithm to account for need of no more than two partial jobs per period; this version is presented in Appendix 1.

7.2 Low level Algorithm

In this section, we detail the scheduling algorithm that solves (6.11) subject to (6.12) and (6.13). This algorithm schedules parts in real-time and can be used along with simulation of failure and repair events.

If N = {1,2,...,n}, we define an ordering function O, O : N → N, with $a_{o(i)} \geq a_{o(j)}$ $\forall i < j$, $i,j \in N$. Where : $a_i = \bar{c}_i / \tau_i$. Ordering of parts by the ratio a_i is an equivalent to the shortest processing time ordering, weighted appropriately by the holding cost.

```
for k = 1 to w do
    repeat
        if M is operational and idle :
            i = find_next_part(k,delay);
            if(i ≠ none & delay = 0)
                schedule p_i immediately; wait until part is completed;
            if(i ≠ none & delay ≠ 0)
                schedule p_i after delay; wait until part is completed;
            if(i = none)
                system idle; wait until end of the period;
        else
            wait till machine repair is complete;
    until(i = none);

find_next_part(k,delay)
    delay = 0;
    /*  Satisfy immediate demand  */
    for i = 1 to n do
        j = O(i);
        if(s(j,k) > 0)   /*  see equation 6.9 for definition of s(j,k)  */
            return j;
    /*  Verify high level flow plan  */

    if  Σ_{i | p_i ∈ f_j} y(i,k) = u(j,k)   for i = 1 to n
            return none;
    /*  Try to verify high level flow plan  */
```

```
for r = k+1 to w do
        for i = 1 to n do

                if u(j,k) -   Σ_{i | p_i ∈ f_j} y(i,k) ≥ 1
                        if d(i,r) > 0   return j;
/*      Only one partial part left - schedule with(out) delay      */
for r = k+1 to w do
        for i = 1 to n do

                if u(j,k) -   Σ_{i | p_i ∈ f_j} y(i,k) ≥ 0
                        if d(i,r) > 0
                                if τ_i > remaining time in period k
                                        return i
                                else
                                        delay = remaining time in period k - τ_i - epsilon
                                        return i;
/*      Default      */
return none;
```

8 Comparison

This section is devoted to the comparison between the performance of the hierarchical model and the branch and bound algorithm. At first, test examples were created; data related to sets of 10 parts each were constructed. For each example, a set of random demands was generated using different starting seeds. The generation of this demand is detailed as follows.

As explained in section 2, n_i, represents the average production per unit time for part p_i. We introduce an integer Q_i indicating the maximal size of demand that can arise for part p_i in an elementary period. Now for each elementary period, we decide to generate a demand for p_i with a probability μ_i, $i = 1,2,\ldots,n$. Then, the size of the demand is chosen, with equal probability, among the set $\{1,2,\ldots,Q_i\}$ of integer values. In other words, once we decide to generate a demand, we determine the size of the demand by the discrete $U(1,Q_i)$ distribution. The process is repeated for each part type and then for each elementary period of the horizon H, i.e., z periods. The probabilities of generating a demand μ_i, can be calculated as follows :

$$\mu_i = \frac{2\, n_i \times \Delta}{(Q_i + 1)}; \qquad i = 1,2,\ldots,n \tag{8.1}$$

Thus, using different seeds, we can generate different sets of demand for period H. Generation of demand in this fashion will ensure the close conformance of the simulated mean demand to the given mean demand (n_i) in the long run.

Since the number of parts considered are very small, reduction of dimensionality by aggregating parts at the high level of the hierarchical model was not considered necessary. Moreover, the demand is deterministic, so the other benifits of aggregation cannot be exploited. Thus, at the high level each family represents a single part. Note that, the function of the low level is merely to prioritize parts within a period, and cannot change the value of the objective function.

All comparisons were performed in a deterministic environment. Furthermore, for the first set, demand was considered for only 16 elementary periods, and the horizon was not rolled. This was with the intent to demonstrate the performance of the hierarchical approach in random isolated cases. It is not difficult to appreciate that the performance of the hierarchical model with rolling horizon will be better owing to its continual "look ahead" nature, while the optimal algorithm is intended to solve the problem horizon after horizon. This constitutes the second set of comparisons. Also, in an unreliable environment, the performance of hierarchical model is expected to be relatively good owing to the incorporation of averages at the high level, opposed to the need for repeated computation of the optimal solution after every repair event in the monolithic case. However, this does not constitute the contents of the present work. In other words, we wish to study a simple set of problems in a deterministic environment that may not necessarily be the most favorable to the hierarchical model.

Table 8.1 presents the characteristics of the part related data. Examples 1 through 4 are the ones for which our assumption related to $c_i^- >> c_i^+ \ \forall i$, is verified. For examples 5 through 9, $c_i^- > c_i^+$ in general, but this is not necessarily true. Finally, for example 10, $c_i^- << c_i^+$, which is absolutely contrary to our assumption. Through these examples, our objectives are to study the performance under the conditions that verify our assumption, and then study the decay when the assumptions cease to hold true.

Example	n	Earliness Penalties	Tardiness Penalties	Average Demand	Maximal Demand	Processing time
1	10	U(1,10)	U(100,500)	U(.15,.25)	2	U(0.2,0.5)
2	10	U(1,10)	U(100,500)	U(2,3)	5	U(0.02,0.05)
3	10	U(1,10)	U(100,500)	U(.15,.25)	4	U(0.05,0.1)
4	10	U(1,10)	U(100,500)	U(.3,.5)	3	U(0.1,0.3)
5	10	U(10,60)	U(50,100)	U(.1,.2)	2	U(0.2,0.5)
6	10	U(10,60)	U(50,100)	U(.1,.2)	2	U(0.2,0.5)
7	10	U(10,60)	U(50,100)	U(.3,.5)	3	U(0.1,0.3)
8	10	U(10,60)	U(50,100)	U(.15,.25)	2	U(0.2,0.5)
9	10	U(10,60)	U(50,100)	U(2,3)	5	U(0.02,0.05)
10	10	U(100,500)	U(1,10)	U(.3,.5)	3	U(0.1,0.3)

Table 8.1 : Part related data for 10 examples

The branch and bound (B&B) algorithm, as well as the algorithms for the hierarchical model (HM) presented in section 7 were coded in C and executed on the SUN SPARC/Unix station 330 platform. Table 8.2 presents the summary of results relating to the value of the objective function and the processing times reported to the nearest second. The branch and bound algorithm was of course not able to determine (or verify) the optimal in all cases, in these cases the program was terminated after about 1 hour, or because of memory constraints. We report the best cost obtained in these cases and indicate it with an asterisk (*); for these cases, processing times are not reported.

To summarize the comparison, we divide the results into three categories depending on the part penalties. The first category belongs to examples 1-4 ($c_i^- >> c_i^+ \ \forall i$), which are inline with our assumptions. For this category, the HM was able to obtain the optimal (or the best solution obtained by B&B) is about 60% of the cases. For 75% of the cases the HM obtained a result within 5% of the optimal value. Solutions were 8% worse than the optimum on an average. Furthermore, it is important to indicate that the HM always found a solution better (if not the same) than the best upper bound heuristic (section 2.1).

The second category belongs to examples 5-9 ($c_i^- > c_i^+$ in general, but, not always), which are a fair deviation from our assumptions. For this category, the HM was able to obtain the optimal (or the best solution obtained by B&B) is about 20% of the cases. For 40% of the cases the HM obtained a result within 10% of the optimal value.

Solutions were 20% worse than the optimum on an average. This decay in performance can be expected. Finally the last category is of example 10 ($c_i^- << c_i^+ \ \forall i$), which is absolutely contrary to our assumptions. For this category, the HM obtained results very far from the optimum. For this case it would be perhaps be better to design a hierarchy with a no holding constraint at the high level with an objective of minimizing the backlogging cost.

Example	case	Number of Jobs	Seed	Branch and Bound		Hierarchical Model	
				Cost	Time	Cost	Time
1	a	33	1234567	17.28	3.73	17.28	0.06
	b	35	999999	102.87*	--	102.57	0.08
	c	21	11111	7.28	0.05	9.85	0.05
2	a	400	1234567	19.35	588.74	19.35	0.15
	b	416	999999	139.83*	--	139.83	0.15
	c	383	11111	47.32*	--	46.03	0.12
3	a	166	1234567	9.44	2.33	10.88	0.10
	b	171	999999	38.85*	--	54.12	0.10
	c	186	11111	50.29*	--	54.59	0.10
4	a	60	1234567	39.13*	--	39.13	0.07
	b	52	11111	9.84	0.33	9.84	0.08
	c	54	55555	109.93*	--	112.78	0.07
5	a	27	1234567	98.60	0.38	133.86	0.07
	b	27	999999	201.29	2.21	250.21	0.08
6	a	32	1234567	31.51	0.12	46.47	0.07
	b	30	99999	115.79	0.18	115.79	0.07
7	a	60	1234567	152.02	295.80	179.09	0.07
	b	52	11111	79.60	0.25	86.52	0.07
8	a	35	1234567	100.85	6.58	100.85	0.08
	b	36	999999	536.40*	--	590.79	0.08
9	a	400	1234567	183.05*	--	212.98	0.13
	b	416	999999	819.53*	--	1308.09	0.13
10	a	60	1234567	45.82*	--	2195.14	0.07
	b	52	11111	9.84	0.23	871.77	0.08
	c	54	55555	103.84*	--	2943.31	0.07

Table 8.2 : Summary of results for 25 demand samples

The second set of comparisons is performed on relatively longer time horizons. We consider a total of 160 elementary periods. The B&B had a horizon of 16 periods (z=16), so 10 problems were solved period by period. The HM had a similar high level horizon (z=16), with a low level horizon of 4 periods. Thus, the horizon at high level was rolled every 4 elementary period. This required the high level problem to be solved 40 times. The results pertaining to these runs are presented in table 8.3.

Example	case	No. of horizons	Seed	Branch and Bound		Hierarchical Model	
				Cost	Time	Cost	Time
1	a	10	1234567	585.22	182.16	143.51	1.85
	b	10	999999	*	--	102.57	0.08
	c	10	11111	107.62	153.63	115.88	1.91

Table 8.3 : Summary of results for 25 demand samples

9 Conclusions

Although the number of examples tried were limited, we can draw the following conclusions. When our assumptions related to the tardiness penalty being much greater than the earliness penalty holds, the hierarchical model seems to result in acceptable solutions. It requires extremely low cpu time and it makes it amenable for application to more complex job-shop problems. It can be demonstrated with similar numerical tests that in an unreliable environment the hierarchical approach continues to perform well. Unfortunately, its performance decays when the assumptions cease to hold true.

References

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

[2] 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.

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

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

[5] 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.

[6] 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.

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

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

[9] 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.

[10] Libosvar, C (1988) : "Hierarchical Production Management : The Flow-Control Layer," Ph.D. Thesis (l'university de Metz), 1988.

[11] 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-, 1984.

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

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

Appendix 1

Modified algorithm 7.1.

```
for k = z down to 1 do
    Initialize :
    u(j,k) = 0 for j = 1;2,...,f;
    Available_capacity := MTBF/(MTBF+MTTR) × Δ;
    for i = 1 to f do
        j = O(i);
        load = θ_j × q(j,k);
        if(load ≤ Available_capacity) then
            Available_capacity = Available_capacity - load;
            u(j,k) = u(j,k) + q(j,k)
        else
            load = (Available_capacity/load) × q(j,k);
            u(j,k) = u(j,k) + load;
            if(k = 1) then
                "Infeasible Plan"; exit;
            u(j,k-1) = ⌈load⌉ - load;
            q(j,k-1) = q(j,k-1) + (q(j,k) - ⌈load⌉)
            for s = i+1 to f do
                j = O(s);
                q(j,k-1) = q(j,k-1) + q(j,k);
            end;
    end;
end.
```