Capacity-Driven Acceptance of Customer Orders for a Multi-Stage Batch Manufacturing System: Models and Algorithms

Robin Roundy Dietrich Chen Pan Chen Metin Çakanyildirim
Michael B. Freimer Vardges Melkonian*

Abstract

An automotive parts manufacturer produces a wide variety of parts in a job-shop environment. Many of the manufacturing operations have substantial setups. When a client phones in an order, the manufacturer must decide quickly whether or not it has the capacity required to accept the order. We develop a simplified formulation of the order acceptance problem. We formulate the discrete-time version as an integer program. The problem is NP-hard, but in 60 out of 60 small test problems the LP relaxation is tight. For larger problems we test several heuristics. Three of the heuristics look promising – simulated annealing, a genetic algorithm, and an LP-based heuristic.

Two strong trends affecting both manufacturing companies and software vendors are *Finite-Capacity Scheduling* (FCS) systems and *Available-to-Promise* (ATP) systems. FCS systems produce and maintain detailed, capacity-feasible schedules of manufacturing facilities. ATP systems quickly evaluate incoming customer orders. They typically verify if the orders can be accepted. If not, they typically offer a smaller order quantity, or offer a later delivery date. The first ATP systems allocated finished-goods inventory and planned receipts to clients, but did not alter production schedules. The idea of combining ATP and FCS systems by altering the production schedule in response to incoming customer orders is gaining in popularity, in spite of the computational difficulty of scheduling problems and the need for very short response times. In this paper, we develop a model that combines many of the features of FCS and ATP systems, and we construct and test viable algorithms for this model.

A European company manufactures a wide variety of components for the automotive industry. Some of their clients place orders for a single shipment of a product at a time. Others call to negotiate for a

*all at School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY 14853-3801. {robin, didi, pchen, metin, mfreimer, vardges}@orie.cornell.edu

sequence of just-in-time shipments, often in small quantities, covering a relatively long period of time. Due to the variety of parts being produced and the volumes in which they are sold, the manufacturing system is characterized by substantial setups, by unstructured multi-stage routings, and by the assembly of multiple manufactured components.

The company has a policy of storing finished goods inventory when necessary, and minimizing work in process (WIP). They accomplish this as follows. The company pays close attention to capacity when loading the manufacturing facility. Lot-for-lot batch sizing is used. Production batches maintain their identity as they progress through the shop. They assign due dates to each manufacturing operation by offsetting the batch due dates by relatively short manufacturing lead times. Finally, they insist that the shop floor adheres to due-date-driven sequencing, and use overtime as needed to keep up.

We develop a model and study algorithms that determine whether or not we can accept a new order without failing to meet our existing commitments. We assume that a detailed, feasible schedule exists in computer memory. When a new order comes in, we attempt to insert it into the current schedule. The current schedule can be modified to accommodate the incoming orders. However, we cannot violate the operation due date constraints. A new order consists of multiple shipments, and we may use one or more production batches to fill it. Thus, lot sizing decisions are made as well.

We develop a Mixed Integer Linear Programming (MILP) formulation of the order insertion problem that bypasses detailed Gantt Chart manipulations but guarantees feasibility. We formulate the problem as a network flow problem with integer variables and side constraints. The problem is proven to be NP-hard in Appendix A, but clients need a quick response. Thus, heuristics are called for. We propose heuristics based on genetic algorithms, tabu search, simulated annealing, myopic search, and shortest-path computations, and report on extensive computational tests.

1 Literature Review

In this section we review portions of the existing FCS, ATP, and Order Acceptance literature. There is a specific branch of FCS problems that involves optimization criteria based on job due dates. Such problems can be roughly divided into the following three types:

(i) For given due dates and capacities, minimize a due-date-related measure. Tardiness *is* allowed.

(ii) Evaluate the feasibility of given due dates. Capacities may or may not be adjusted. Tardiness *is not* allowed.

(iii) For given capacity levels, quote due dates and/or reject orders. Accepted orders are usually delivered on time.

A good portion of classical job shop scheduling theory, e.g. Conway, Maxwell, and Miller (1967) and Baker (1974), deals with problems of type (i). Roughly speaking, job shop scheduling approaches are branch and bound inspired towards finding either bounds – as in Lawler, Lenstra, Kan, and Shmoys (1993) – or heuristics. A survey of job shop scheduling can be found in Blazewicz, Domschke, and Pesch (1996).

In this paper, our focus is on heuristics. Among heuristic approaches, the shifting bottleneck procedure as employed by Adams, Balas, and Zawack (1988) stands out for successfully merging analytical techniques with heuristic ideas. This iterative procedure focuses on the bottleneck machine and solves a single machine problem at each iteration. Schedules on the other machines are geared towards the schedule of the bottleneck machine. van Laarhoven, Aarts, and Lenstra (1992) use simulated annealing to obtain an approximate solution to the job shop scheduling problem. They aim to minimize the makespan. Two solutions are defined to be neighbors if it is possible to obtain one from the other by switching the sequence of two operations which contribute to the makespan. It is shown that their method converges to the global minimum asymptotically. The paper also numerically compares simulated annealing with the shifting bottleneck procedure. Kim and Kim (1995) minimize the weighted sum of tardiness and earliness after aggregating all manufacturing operations into two categories: machining and assembly. They use both simulated annealing and genetic algorithms, and compare them numerically. Enns (1996) examines blocked time and event driven algorithms for FCS in terms of their flow time, schedule stability, and delivery performance. He concludes that event driven algorithms, when due dates are specified internally, yield the best results.

Problems of type (ii) combine short term capacity planning with scheduling. The easiest way of tweaking capacity in the short term is by using overtime. Both Gelders and Kleindorfer (1974) and Holloway and Nelson (1974) adopt this view and combine scheduling with overtime decisions. The former considers a one machine problem with given jobs which are never rejected. The criterion is a combination of overtime costs plus flow time and tardiness measures. The principal decision variable is the amount of overtime planned at the end of a regular time period. Also see Gelders and Kleindorfer (1975). Holloway and Nelson (1974) insert overtimes into the regular schedule to cut down tardiness. They also generate tardiness vs. overtime trade-off curves.

A variant of problems in category (ii) in the cyclic scheduling realm is studied by Loerch (1990) and Loerch and Muckstadt (1994). First, for a given fixed cyclic schedule, batch sizes per cycle are calculated based on constant cycle demands. Then, the longest path is found in a network, where processes and

processing times are represented by nodes and arcs, respectively. Depending on the length of that path, the schedule is declared feasible or infeasible. In case of infeasibility, overtime and outsourcing alternatives are investigated through a linear program.

A survey of works concerning (iii) can be found in Cheng and Gupta (1989). A version of the due date quotation problem with stochastic arrival times is given by Bookbinder and Noor (1985). More recently, the due date quotation problem is studied in Duenyas and Hopp (1995) and Duenyas (1995). The former establishes the structure of due date quotation policies under various modeling assumptions: infinite capacity and finite capacity with fixed or variable lead times. In general, the longer the quoted lead time, the smaller the probability that a customer places an order. In the case of finite capacity with fixed lead times, the issue is whether an order, with a given lead time, can be accepted or not. The structural results are of the following form: accept the new order if there are fewer than a particular number of orders already in process. It is also shown that the quoted lead times increase with the number of orders in the process. Duenyas (1995) extends Duenyas and Hopp (1995) for multiple customer classes. Each customer class is assumed to be demanding the same product. They differ, however, in terms of price and lead time preferences. Heuristics are provided for making the type of decisions in Duenyas and Hopp (1995). In a deterministic setting, Luss and Rosenwein (1993) provide a due date quotation model which minimizes weighted positive deviations between quoted due dates and given ones. They propose a Lagrangean relaxation based heuristic which takes good candidate schedules as input. In a similar study, Enns (1998) works on lead time selection for good delivery performance.

In practice, jobs are dynamically scheduled and re-scheduled over and over as circumstances change with new jobs, failing machines, etc. Thus, at a given time, there are both scheduled but uncompleted and unscheduled jobs. Due dates for unscheduled jobs are more flexible than due dates of the scheduled jobs. Thus, the problem is to insert new jobs into the existing schedule without violating the due dates of the scheduled jobs. Scheduling theory is generally geared towards tackling a static instance as opposed to considering dynamically inserting jobs in an existing schedule. This is a big gap between practice and theory. This paper attempts to partially fill that gap.

Papers mentioned above with the exception of Duenyas and Hopp (1995) do not address the job insertion issue. One way to insert jobs is using a variation of *Material Requirements Planning*, called MRP-C – as in Hopp and Spearman (1996). MRP-C checks for the feasibility of capacity while scheduling lots to periods for given due dates. It plans production by netting out work-in-process inventory and by curtailing desired production levels in accordance with capacity. Altering capacity levels and/or pushing due dates are both viable options in case of an infeasible schedule. Another job insertion approach, within the same realm of

type (ii) problems, is given by Akkan (1996) in which he considers finding a minimum cost overtime plan. His insertion procedure does not increase the completion time or decrease the starting time of a job. A shortest path formulation is given but heuristics are employed for large problems.

Existing ATP literature is limited. Initial applications of ATP systems simply compared demand against inventories for order acceptance/rejection decisions. Recent applications of popular commercial software, however, alter the production schedule: either with a long-time-bucket approach like MRP, or with a short-time-bucket approach by communicating with an FCS module. Thus, they close the gap between order acceptance and manufacturing functions. These changes are made possible by the introduction of information technologies (IT) into the manufacturing industry. King (1996) and Hill (1998) discuss ATP applications at Unisys Corp. and for the apparel industry, respectively. Gould (1998) and Taylor and Bolander (1997) touch upon the use of ATP in the automotive industry and flow manufacturing industries (such as chemical, food, pharmaceutical), respectively. ATP is not simply a consequence of IT; it is an outcome of the evolution of factory management philosophies. The shift in these philosophies from a resource-centric point of view toward a customer-centric point of view is discussed in Layden (1996) and Kevin (1996). They also point out that methods for integrating ATP and order scheduling modules like FCS will be a central tool in the implementation of these new philosophies.

A body of literature, closely related to ATP, has recently surfaced: *Order Acceptance*. The issue in *Order Acceptance* is adding a given customer order to a pre-existing schedule. An example of *Order Acceptance* work can be seen in Slotnick and Morton (1996) in which late orders are allowed. It can hence be classified as a problem of type (i). Each customer order has a revenue and linear lateness penalty. The job selection problem is to maximize the sum of revenues minus the sum of lateness penalties over the subset of the orders selected. A branch and bound algorithm and two heuristics are given. Heuristics rely on a method of detecting orders that will be selected by preprocessing. On the other hand, Wester, Wijngaard, and Zijm (1992) do not allow late orders, thus their work is an example of category (iii). They study order acceptance and scheduling strategies in a single machine and production-to-order environment. Three heuristics are proposed. The first one, a monolithic approach, uses all the details of an existing schedule. The second and third both use a total workload based criterion for order acceptance. The third differs in scheduling in that it employs a myopic rule. Computational experiments and insights on heuristic performances are provided as well. In a related work, Dauzere-Peres and Lasserre (1997) jointly study lot-streaming and job shop scheduling. However, they consider scheduling all jobs at once. Their approach is based on iterating between lot-streaming and scheduling until the makespan is decreased sufficiently close to its lower bound. When setup times are negligible, their procedure yields makespans approaching the lower bounds even with

a small number of lots. For a comparable makespan performance in the case of considerably large set up times, more lots are needed.

Our paper is also an *Order Acceptance* study related to category (iii) and can be considered as an extension of Wester, Wijngaard, and Zijm (1992) to the case of multiple machines. Our paper also differs in that lot sizes are decision variables. We structure our paper as follows: in section 2, we will state some key assumptions and define the notation used in the paper. In section 3, we will focus on mathematical characterization of feasible schedules. Our MILP formulation is given in section 4. Various heuristics are discussed in section 5. In section 6, we will examine the computational results.

2 Assumptions and Notation

In order to mathematically model the problem described in the introduction, we make a number of assumptions.

- Each order involves only one type of product but may consist of several shipments, each of which has a specified shipment quantity and a specified shipment date.
- Each product is associated with a multi-stage product routing, which may involve assembly operations. Each operation must be performed on a given machine. For a given product, each machine appears in the routing at most once.
- For each operation, there is a deterministic setup time and a setup cost, a per-part processing time on the machine.
- Each machine has a nominal lead time. Due dates for batches are offset by nominal lead times to obtain a due date for each operation. We do not schedule an operation to start more than one nominal lead time before its due date.
- Machines are operated continuously. Overtime and other means of changing capacity are not modeled.
- The current production schedule is feasible. An incoming order is accepted only if a feasible schedule that includes it can be found. Among all feasible schedules, the one that minimizes the sum of all setup costs and holding costs is preferred.

We now introduce some notation. The planning horizon spans the time interval $[0, Z]$. A client order consists of a product j and a collection of shipments $i = 1, \ldots, s$. For each shipment i, quantity $Q(i)$ must be shipped out by date $D(i)$.

The company is to satisfy the client order with a number of production batches indexed by b, $1 \leq b \leq B$. For each batch b, quantity $q(b)$ is assigned a due date $\tau(b)$. Product j requres M operations on M distinct machines. Thus a total of BM operations is required. Operation k is done on machine $m(k)$ with a setup time $s(k)$, a nominal lead time $L(k)$, and a processing time per part $tpp(k)$. Thus, the processing time of operation k is $p(k) = s(k) + tpp(k) \cdot q(b)$ if k corresponds to batch b. At any given time, at most one operation can be scheduled on any machine. The collection of operations associated with the client orders that have already been accepted and the associated data constitute a *production plan.*

3 Characterizations of Feasible Schedules

In this section, we focus on the question of whether or not a production plan is feasible, and what flexibility exists for inserting a new customer order into a feasible production plan. Our definition of feasibility includes the requirement that each operation k on machine m must finish by its due date $d(k)$, and start at most $L(k)$ before that. Consequently, we can answer these questions by considering each machine in isolation.

In this section, we consider only those operations that are processed on a single machine m. We assume without loss of generality that these operations are indexed from 1 to J, and that $d(k) \leq d(k+1)$. The set of operations $\{d(k), p(k) : 1 \leq k \leq J\}$ is called the *current workload on machine m.*

For convenience, we write $L := L(m)$ and $T(k) := \sum_{j=1}^{k} p(j)$. For completeness, we define $d(0) = 0, d(J+1) = Z + L, T(0) = 0, T(-1) = -L, T(J+1) = T(J)$.

In view of the fact that for each operation, the due date and the earliest start date differ by L, we have the following *Earliest-Due-Date* (EDD) result. It easily proven using adjacent pairwise interchange arguments.

Lemma 1 *A feasible schedule exists for machine m if and only if there is a a feasible schedule that sequences the operations in EDD order, i.e., in the order* $1, 2, 3, \ldots$

Lemma 1 tells us how to sequence the operations, but it does not give start times or completion times. We define a *schedule* for machine m to be a function $\sigma(\delta)$, defined for $\delta \in [0, Z]$, with the following properties:

$$\sigma(0) = 0 \tag{1}$$

$$\sigma(\cdot) \quad \text{is non-decreasing} \tag{2}$$

$$\sigma(\delta + \gamma) \leq \sigma(\delta) + \gamma \qquad \forall\, \delta \geq 0, \quad \gamma > 0. \tag{3}$$

Note that any function which satisfies (1)-(3) is continuous. Thus the function

$$\sigma^{-1}(t) := \sup\{\delta : \sigma(\delta) = t\} \tag{4}$$

is a well-defined right-continuous non-decreasing function. Let $\sigma^{-1}(t^-) := \lim_{\epsilon \searrow 0} \sigma^{-1}(t-\epsilon)$. Then

$$\sigma(\sigma^{-1}(t)) = t \quad \text{and} \quad \sigma^{-1}(\sigma(\delta)) \geq \delta \geq \sigma^{-1}(\sigma(\delta)^-) \quad \forall\ \delta. \tag{5}$$

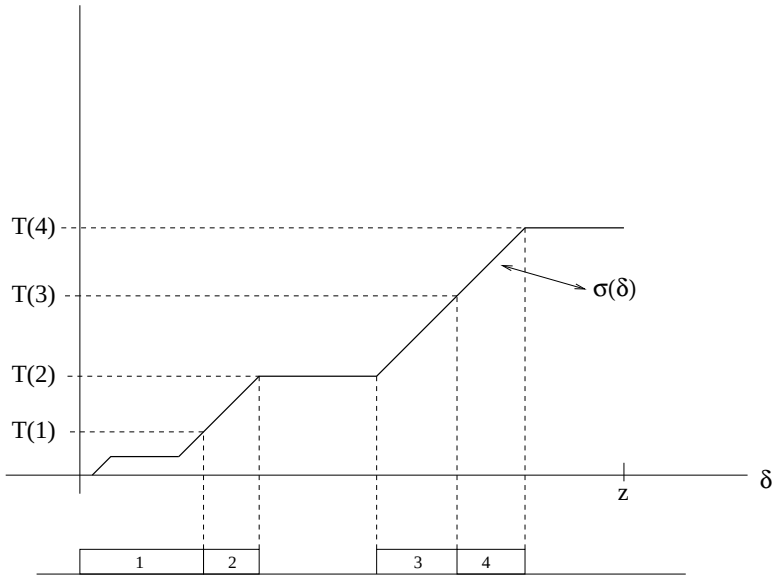


Figure 1: A Schedule

A schedule is converted into a Gantt Chart as follows (see Figure 1). The machine works on operation k from time $\sigma^{-1}(T(k-1))$ to time $\sigma^{-1}(T(k)^-)$. If $\sigma'(\delta)$ exists then it is the rate at which the machine is working at time δ. In particular, suppose that $\sigma(\delta) = t \quad \forall\ \delta \in [a,b]$. Then $\sigma'(\delta) = 0 \quad \forall\ \delta \in [a,b]$ as demonstrated in Figure 1. If $T(k-1) < t < T(k)$ then $(b-a)$ units of idle time are inserted part of the way through the processing of operation k. If $t = T(k)$ then at least $(b-a)$ units of idle time are inserted in the schedule after operation k finishes and before operation $k+1$ starts.

Our assumptions on $\sigma(\cdot)$ imply that the machine is allowed to work at any speed between 0 and 1, and that the speed can be changed at any time. More restrictive assumptions could be made without impacting any of our main results.

A schedule $\sigma(\cdot)$ is *feasible* if

$$d(k) \geq \sigma^{-1}(T(k)^-) \quad \text{and} \quad \sigma^{-1}(T(k-1)) \geq d(k) - L \quad \forall\ 0 \leq k \leq J+1 \tag{6}$$

or equivalently,

$$\sigma(d(k)) \geq T(k) \quad \text{and} \quad \sigma(d(k) - L) \leq T(k-1) \quad \forall\ 0 \leq k \tag{7}$$

The second inequality of (6) states that operation k cannot start before time $d(k)-L$. The first inequality implies that operation k must finish by time $d(k)$. Clearly, the schedule $\sigma(\cdot)$ is feasible for machine m if and only if (6) holds. Equivalence of (6) and (7) follows from (5).

The following lemma gives a direct characterization of the data sets $(d(\cdot), T(\cdot), L)$ which admit a feasible schedule.

Lemma 2 *A feasible schedule exists if and only if (ii) holds for all j, k that satisfy $0 \leq j \leq J, 1 \leq k \leq J$ and (i).*

$$\textit{(i)} \quad d(j) - L \leq d(k) \tag{8}$$

$$\textit{(ii)} \quad T(k) + (d(j) - L - d(k)) \leq T(j-1) \tag{9}$$

Proof. Assume that $\sigma(\cdot)$ is feasible and that $d(j) - L \leq d(k)$. $T(k) \leq \sigma(d(k)) + T(j-1) - \sigma(d(j) - L) \leq T(j-1) + d(k) - d(j) + L$, where the first inequality follows from (7) and the second from (1)-(3). This proves necessity. For sufficiency, see the proof of Lemma 3 below. ■

Note that setting $j = 0$ in (9) we obtain

$$T(k) \leq d(k) \quad \forall\, k. \tag{10}$$

Condition (10) states that there must be adequate capacity to complete the first k operations before the deadline of the k-th operation. If $d(j) \leq d(k)$ then in the time interval $[d(j), d(k)]$, operations requiring a total of $T(k) - T(j-1)$ units of processing time are due. Processing must be completed by time $d(k)$ and cannot begin before time $d(j) - L$, so $d(k) - d(j) + L$ units of time are available to do the work. This leads to (9). Also note that by setting $k = j$ in (9) we obtain

$$p(k) \leq L. \tag{11}$$

We define the following notation,

$$\begin{aligned}
x \vee y &:= \max\{x, y\} \\
x \wedge y &:= \min\{x, y\} \\
x^+ &:= x \vee 0 \\
\mathcal{L}'(k, \delta) &:= [T(k) - (d(k) - \delta)^+] \qquad \forall\, 0 \leq \delta \leq Z + L,\ \forall\, 0 \leq k \leq J+1 \\
\mathcal{E}'(k, \delta) &:= [T(k-1) + (\delta + L - d(k))^+] \quad \forall\, -L \leq \delta \leq Z,\ 0 \leq k \leq J+1.
\end{aligned}$$

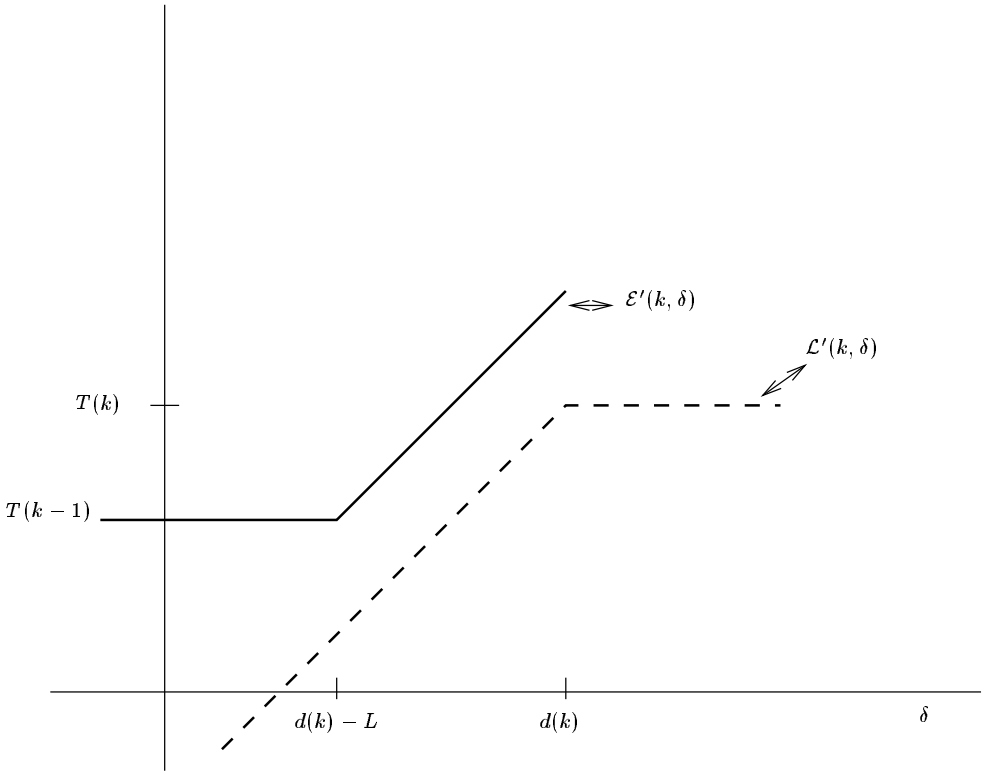


Figure 2: Constructing Earliest and Latest Schedules

Note that $\mathcal{L}'(0,\delta) = 0$, $\mathcal{E}'(0,\delta) = \delta$, and $\mathcal{L}'(J+1,\delta) = T(J)$. We define the *latest feasible schedule* as follows (see Figure 2):

$$\mathcal{L}(\delta) := \max_{k:0\le k\le J+1}\{\mathcal{L}'(k,\delta)\} = \max_{d(k)\le\delta}\{T(k)\} \vee \max_{d(k)\ge\delta}\{T(k)-(d(k)-\delta)\},\ 0\le\delta\le Z+L. \tag{12}$$

Similarly, we define the *earliest feasible schedule* as

$$\mathcal{E}(\delta) := \min_{k:1\le k\le J+1}\{\mathcal{E}'(k,\delta)\} = \min_{d(k)-L\le\delta}\{\delta-d(k)+L+T(k-1)\} \wedge \min_{d(k)-L\ge\delta}\{T(k-1)\},\ -L\le\delta\le Z. \tag{13}$$

See Figure 2 for a visualization of constructing earliest and latest schedules. Note that $\mathcal{L}(\delta)\ge 0$, $\delta\ge\mathcal{E}(\delta)$, and $T(J)\ge\mathcal{E}(\delta)$.

The intuition behind the definition of $\mathcal{L}(\delta)$ is simple. Let $\sigma(\delta)$ be a feasible schedule. Then $\sigma(d(k))$ is the cumulative production by time $d(k)$, and it must be at least $T(k)$. If $d(k) > \delta$ then $\sigma(\delta)$ must be at least $T(k)-(d(k)-\delta)$. Thus, $\sigma(\delta)\ge\mathcal{L}'(k,\delta)$. Consider $\mathcal{E}(\delta)$ now. Clearly, $\sigma(\delta)\le\delta$. Operation k cannot be started before time $d(k)-L$, so $\sigma(\delta)\le T(k-1)+[\delta-(d(k)-L)]^+$.

Figure 3 illustrates the constraints of $\mathcal{E}$ and $\mathcal{L}$. The graph of the step function $\tau(\delta)$ is constructed by connecting the points

$$\{(d(0),T(-1)),(d(0),T(0)),(d(1),T(0)),(d(1),T(1)),\ldots,(d(2),T(1)),\ldots,(d(J+1),T(J+1))\}.$$

Note that (7) implies $\tau(\delta + L) \geq \mathcal{E}(\delta)$ and $\mathcal{L}(\delta) \geq \tau(\delta) \quad \forall\ \delta$. Furthermore, note that $\tau(0) = \tau(-L)$ and $\tau(Z) = \tau(Z + L)$, so the solid and dashed curves in Figure 3 touch at $\delta = 0$ and $\delta = Z$.

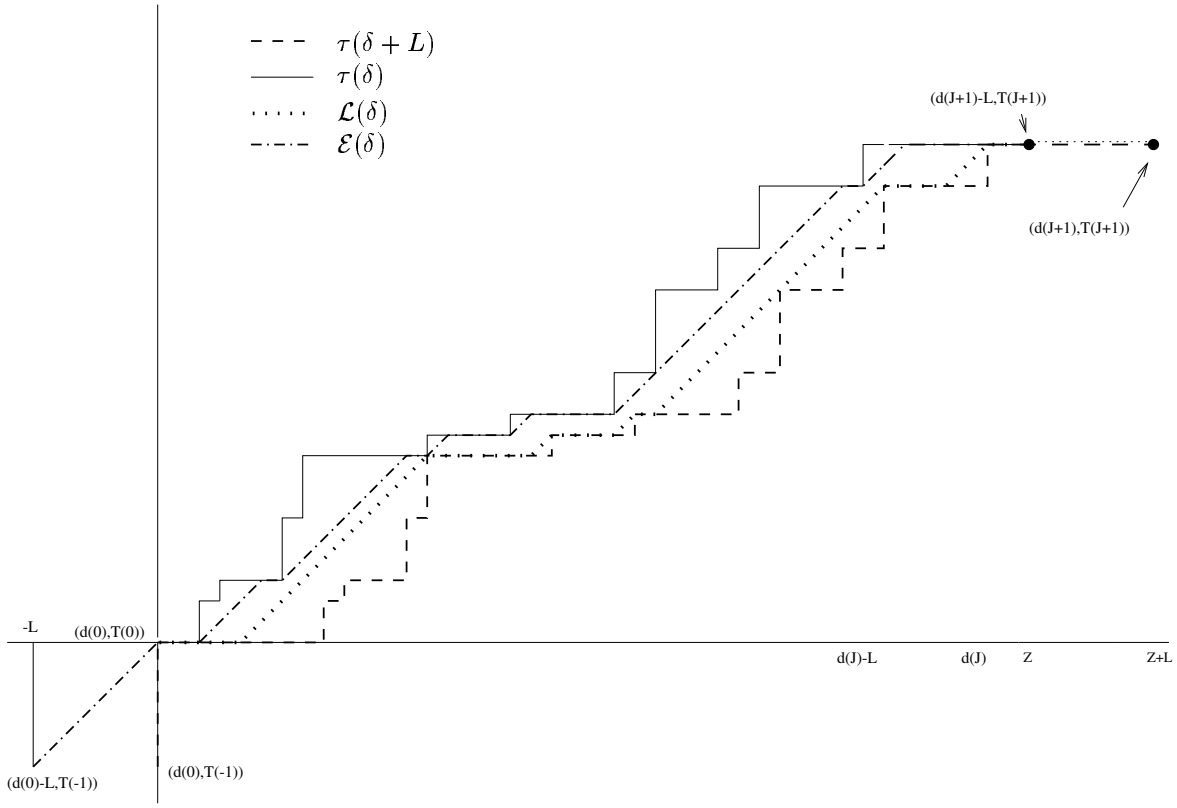


Figure 3: The Earliest Schedule $\mathcal{E}$ and the Latest Schedule $\mathcal{L}$

Lemma 3 *If Lemma 2 holds then $\mathcal{L}$ and $\mathcal{E}$ are both feasible schedules, i.e., they satisfy (1)-(3), (6), and (7). Furthermore,*

$$\mathcal{E}(\delta - \mathcal{L}) \leq \mathcal{L}(\delta) \leq \mathcal{E}(\delta). \tag{14}$$

Proof. Clearly, $[\mathcal{L}'(k,\delta) \vee \mathcal{L}'(0,\delta)]$ and $[\mathcal{E}'(k,\delta) \wedge \mathcal{E}'(0,\delta)]$ satisfy (1)-(3). It is easy to verify that maxima (resp., minima) of a set of functions which satisfy (1)-(3) must also satisfy (1)-(3). Thus $\mathcal{L}(\cdot)$ and $\mathcal{E}(\cdot)$ are schedules. We now prove that

$$\mathcal{L}(\delta) \leq \mathcal{E}(\delta) \quad \forall\ \delta. \tag{15}$$

By (12) and (13) it suffices to show that $\mathcal{L}'(k,\delta) \leq \mathcal{E}'(j,\delta) \quad \forall\ j,k,\delta$. The shapes of $\mathcal{L}'(k,\cdot)$ and $\mathcal{E}'(j,\cdot)$ imply that it suffices to show that $T(k) = \mathcal{L}'(k,d(k))$ is less than or equal to $\mathcal{E}'(j,d(k)) = A := [T(j-1) + (d(k) + L - d(j))^+]$. If $d(k) + L - d(j) \leq 0$ then $j > k$, so $T(k) \leq T(j-1) = A$. If $d(k) + L - d(j) \geq 0$ then by Lemma 2, $T(k) \leq T(j-1) + d(k) + L - d(j) = A$. Thus $\mathcal{L}(\delta) \leq \mathcal{E}(\delta)\ \forall\ \delta$.

Setting $\delta = d(k)$ in (12), we see that $T(k) \leq \mathcal{L}(d(k)) \leq \mathcal{E}(d(k))$. Setting $\delta = d(k) - L$ in (13), we see that $T(k-1) \geq \mathcal{E}(d(k) - L) \geq \mathcal{L}(d(k) - L)$. Thus both $\mathcal{L}(\cdot)$ and $\mathcal{E}(\cdot)$ are feasible. It remains to be shown that $\mathcal{E}(\delta) \leq \mathcal{L}(\delta + L)$.

Let $d(k-1) \leq \delta \leq d(k)$. Then $\mathcal{L}(\delta) - \mathcal{E}(\delta - L) \geq \mathcal{L}(k-1, \delta) - \mathcal{E}(k, \delta - L) = T(k-1) - T(k-1) = 0$. ■

Note that by (7), (13), and (15), $T(J) \geq \mathcal{E}(Z) \geq \mathcal{L}(Z) \geq T(J)$. Thus $\mathcal{E}(Z) = \mathcal{L}(Z) = T(J)$. The next lemma justifies our definition of $\mathcal{E}(\cdot)$ and $\mathcal{L}(\cdot)$ as the earliest and latest feasible schedules.

Lemma 4 *If $\sigma(\cdot)$ is a feasible schedule then $\mathcal{L}(\delta) \leq \sigma(\delta) \leq \mathcal{E}(\delta) \quad \forall\, 0 \leq \delta \leq Z$.*

Proof. It suffices to show that $\mathcal{L}'(k, \delta) \leq \sigma(\delta) \leq \mathcal{E}'(k, \delta) \quad \forall\ k, \delta$. Considering (2)-(3) and the shape of $\mathcal{L}'(k, \delta)$ and $\mathcal{E}'(k, \delta)$, it suffices to show that $\sigma(d(k)) \geq \mathcal{L}'(k, d(k)) = T(k)$ and $\sigma(d(k) - L) \leq \mathcal{E}'(k, d(k) - L) = T(k-1)$. This follows from (7). ■

This concludes our study of feasible schedules as well as our characterization of the current workloads $\{d(k), p(k) : 1 \leq k \leq J\}$ that admit feasible schedules. Before addressing order insertion we make a few observations about idle time and unused capacity in feasible schedules.

Let

$$l(t) \quad := \quad t - \mathcal{L}(t) \qquad\qquad 0 \leq t \leq Z \tag{16}$$

$$e(t) \quad := \quad t - L - \mathcal{E}(t - L) \qquad 0 \leq t \leq Z + L. \tag{17}$$

Note that $e(\cdot)$ and $l(\cdot)$ satisfy (1)-(3). Consequently, they are continuous. Also note that $l(0) = 0 = e(\delta)$ for $0 \leq \delta \leq L$, that $e(Z + L) = l(Z) = Z - T(J)$, and that $l(\delta) = \delta - T(J)$ for $Z \leq \delta \leq Z + L$. By Lemma 4, for all feasible schedules $\sigma(\cdot)$,

$$l(t) \geq t - \sigma(t) \geq e(t + L). \quad \forall \quad 0 \leq t \leq L. \tag{18}$$

Thus, $l(t)$ is the maximum amount and $e(t + L)$ is the minimum amount of idle time in the interval $[0, t]$ in any feasible schedule.

Let $l^{-1}(c) = \sup\{t : l(t) \leq c\}$ and $e^{-1}(c) = \inf\{t : e(t + L) \geq c\}$. By (18), in any feasible schedule $\sigma(\cdot)$ and for any c,

$$l^{-1}(c) - \sigma(l^{-1}(c)) \leq l(l^{-1}(c)) = c = e(e^{-1}(c) + L) \leq e^{-1}(c) - \sigma(e^{-1}(c)). \tag{19}$$

The left inequality in (19) is tight if $\sigma(\cdot) = \mathcal{L}(\cdot)$, and the right inequality is tight if $\sigma(\cdot) = \mathcal{E}(\cdot)$. Thus, the total idle time in $[0, t]$ is equal to c for some $t \in [l^{-1}(c), e^{-1}(c)]$.

Note that $l(Z)$ is the total amount of unused capacity in $[0, Z]$. This capacity is available to accomodate incoming orders. However, we must meet the commitments embodied in the current workload on machine m, and these commitments constrain the manner in which this capacity is used.

Conceptually, we define a quantity called the c^{th} *unit of unused capacity*, defined for each c, $0 \leq c \leq l(Z)$. Motivated by (19), we postulate that if the c^{th} unit of capacity is to be used for a new order, it must be used between time $l^{-1}(c)$ and $e^{-1}(c)$. Thus, it must be used on a new batch whose deadline lies between time $l^{-1}(c)$ and $e^{-1}(c) + L$.

As an illustration, suppose that we are thinking about adding a new operation to the current workload. This operation would use the c^{th} unit of unused capacity for each $c \in [c', c' + p']$. The deadline of the new operation is t. According to the previous paragraph, for each $c \in [c', c' + p']$ we want $l^{-1}(c) \leq t \leq e^{-1}(c) + L$. Since $l(\cdot)$ and $e(\cdot)$ are non-decreasing, it suffices to have $l^{-1}(c' + p') \leq t \leq e^{-1}(c') + L$, or equivalently,

$$l(t) \geq c' + p' \quad \text{and} \quad c' \geq e(t). \tag{20}$$

More formally, a *quotation* consists of the operations $k, 1 \leq k \leq J'$, and the associated due date $d'(k)$ and processing time $p'(k)$. We want to determine whether or not machine m has the capacity to add these operations to its current workload. We allocate unused capacity to the operations by specifying *capacity-assignment* numbers $\{c'(k), 1 \leq k \leq J'\}$. The c^{th} unit of unused capacity is allocated to operation k for each $c, c'(k) < c \leq c'(k) + p'(k)$. Following the logic that led to (20), a *feasible quotation* consists of a quotation and a set of capacity-assignment numbers that satisfy

$$e(d'(k)) \leq c'(k), \tag{21}$$

$$c'(k) + p'(k) \leq l(d'(k)), \tag{22}$$

$$c'(k) + p'(k) \leq c'(k+1). \tag{23}$$

The last inequality states that the c^{th} unit of unused capacity cannot be allocated to more than one operation. Figure 4 shows a feasible quotation.

Given a quotation, if there are capacity-assignment numbers that will make the quotation feasible, then the following capacity-assignment numbers work.

$$c'(1) = e(d'(1)) \tag{24}$$

$$c'(k) = e(d'(k)) \vee (c'(k-1) + p'(k-1)), \qquad k > 1. \tag{25}$$

These numbers assign the earliest available capacity to each operation.

Theorem 1 *Assume that the current workload on machine m admits a feasible schedule. Then*

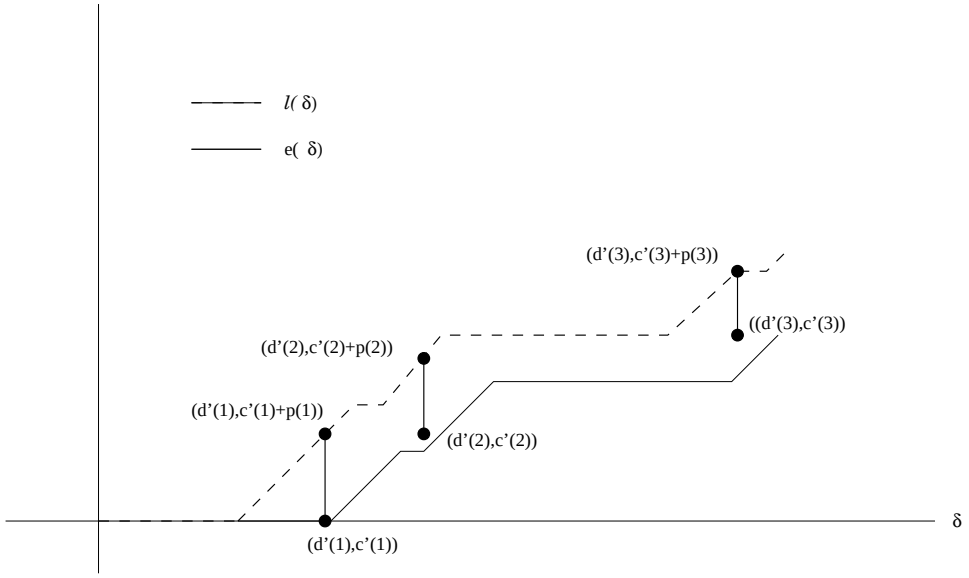


Figure 4: Feasible Quotation

(i) Given a feasible quotation, there is a feasible schedule that completes all of the operations in both the current workload on machine m, and the feasible quotation.

(ii) For a given quotation, assume that there is a feasible schedule that completes all of the operations in both the quotation and the current workload. Then there is a set of capacity assignment numbers that make the quotation feasible.

Proof. See Appendix B. ■

4 Order Insertion

In section 3, time was treated as a continuous variable. If t and all processing times, due dates and nominal lead times are integers, then $e(t)$ and $l(t)$ are also integers. In this section, we provide a network based integer programming formulation for the integer-valued insertion problem.

Assume that the current production plan is feasible, and that we want to insert a new client order for a specific product j into the existing production plan. Recall that our planning horizon is $[0, Z]$ and that orders have due dates $d(j)$. Let m be a generic index for a machine and M be the set of machines that are used to make product j. If m produces the finished product, then $\theta_m = 0$. Otherwise, $\theta_m = L(m\prime) + \theta_{m'}$, where m' is the (unique) successor of m in the routing of product j. Let e_{mt} and l'_{mt} be the discretizations of the corresponding functions $e(t)$ and $l(t)$ of the previous section, and let $l_{mt} = l'_{mt} \wedge l_{mZ}$. Thus $e_{m0} = l_{m0} = 0$

and $e_{m,Z+L} = l_{mt}$ for $Z \leq t \leq Z + L(m)$. Also by (18) $l_{mt} \geq e_{m,t+L}$. Recall that a client order consists of a product j, a collection of shipments i, $1 \leq i \leq s$, and a shipment quantity $Q(i)$ and a shipment date $D(i)$ for each shipment. Suppressing j, we define:

- $D = \sum_{i=1}^{s} D(i)$: the total quantity in the client order.
- e_{mt}: cumulative idle time in the *earliest* schedule on machine m at time $t - L$ (see (18)).
- l_{mt}: cumulative idle time in the *latest* schedule on machine m at time t (see (18)).
- sut_m, tpp_m: setup and processing times for product j on machine m.

We measure setup and holding cost in the following way. In figure 5, the shape of the solid curve is a function of the shipment quantity and time in the client order. Consequently, area B is constant. The shape of the dashed curve is determined by the quantity and timing of the new production batches. The holding cost is proportional to area H. Moreover, the sum of areas A, H, B is constant. Thus, minimizing the holding cost is equivalent to maximizing the sum of the rectangular areas indicated by A. Thus, the cost of producing a batch of q units with due date t can be written as

$$c_{tq} = \left(\sum_{m \in M} S_m \right) - g \cdot t \cdot q, \tag{26}$$

where S_m is the setup cost on machine m and g is the holding cost per time per unit.

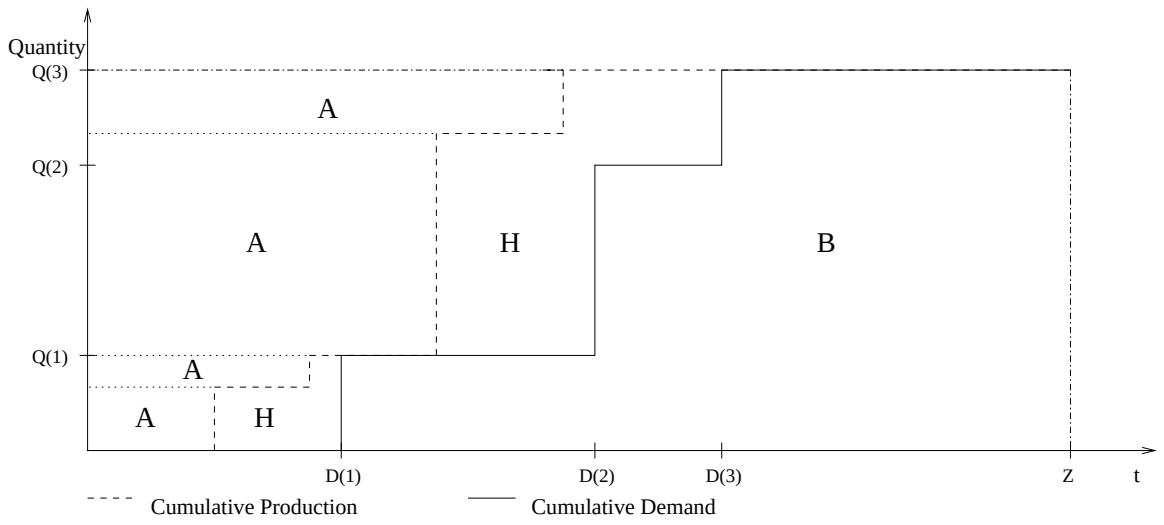


Figure 5: Cumulative Demand vs. Production

The order insertion network for machine m will be denoted by $G_m = (N_m, A_m)$ where $(m, t, h) \in N_m$ if $e_{mt} \leq h \leq l_{mt}$ and $0 \leq t \leq Z + L(m)$. Node $(m, 0, 0)$ is the source and node $(m, Z + L(m), v_m)$ is the sink,

where $v_m := l_{m,Z+L(m)}$. There will be three types of arcs in A_m: production, advance-the-clock, and idle arcs (see figure 6).

- *production arcs*, which are denoted by (m, t, h, q). Traversing this arc corresponds to creating a new batch of q units with due date $t + \theta_m$ by consuming the portion of idle time $(h, h + sut_m + tpp_m \cdot q)$. Naturally, this arc may or may not exist depending on e_{mt} and l_{mt}. It connects nodes $(m, t-1, h)$ and $(m, t, h+sut_m+tpp_m \cdot q)$ for $t \in \{1, \ldots, Z\}$, $h \in \{e_{mt}, \ldots, l_{m,t-1}\}$ and q such that $h+sut_m+tpp_m \cdot q \leq l_{mt}$.

- *advance-the-clock arcs*, which are represented by (m, t, h). Traversing this arc corresponds to advancing to the next time period without using any of the available capacity. It connects (m, t, h) to $(m, t+1, h)$ for $t \in \{0, 1, \ldots, Z + L - 1\}$ and $h \in \{e_{m,t+1}, \ldots, l_{mt}\}$.

- *idle arcs*, which can be represented by (m, t, h). Traversing this arc corresponds to deciding at time t that the h^{th} unit of unused capacity on machine m will not be used for this client order. It connects (m, t, h) to $(m, t, h + 1)$ for $t \in \{1, \ldots, Z\}$ and $h \in \{e_{mt}, \ldots, l_{m,t-1}\}$.

Note that advancing the clock does not consume unused capacity. An example of a production network for ten time periods is depicted in figure 6. In this network, all arcs move from left to right and/or upward. The setup and processing times per part are both one unit. Thus, the shortest production activity requiress at least two units of unused capacity.

It must be emphasized that there is not a production arc from node (2,0) to (3,2) or to (3,3). If these arcs existed they would have to consume the first unit of unused capacity for a batch with due date $3 + \theta_m$ on machine m. However, this is not possible because $e_{m3} = 1$. Furthermore, nothing can be produced in period 10 because only one unit of unused capacity is available.

The outgoing arc from (0,0) and entering arc to (11,6) = (Z + L(m), T(J, m)) are both constrained to have one unit of flow. If flow along a path is 1, then the arcs of this path specify the sizes and due dates of new production batches. For example, the path (0,0), (1,0), (2,0), (2,1), (3,3), (4,3), (5,3), (6,3), (7,3), (8,6), (9,6), (10,6), (11,6) corresponds to creating one production batch with quantity 1, processing time 2, and due date of $3 + \theta_m$, and another batch with quantity 2, processing time 3, and due date $8 + \theta_m$, on machine m.

Since no arc in the production network moves downward, each unit of unused capacity can be used only once. Note that the existence of a path indicates that a feasible production schedule incorporating the new production batches exists (Theorem 1), but it does not determine the schedule.

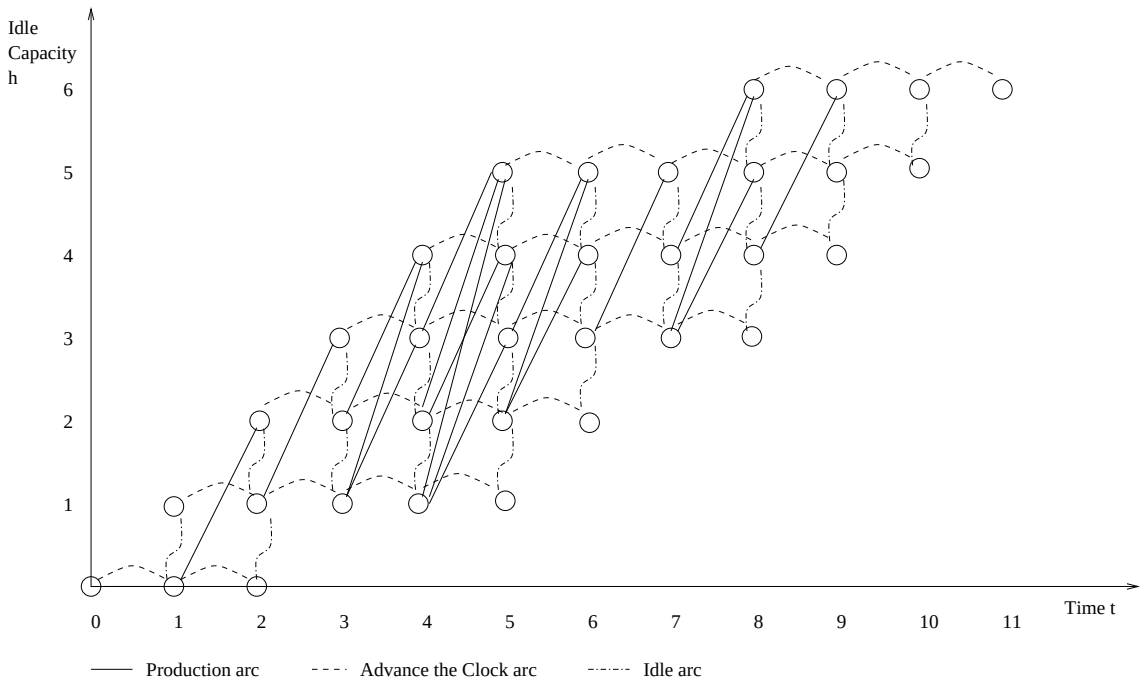


Figure 6: Order Insertion Network

Next, a mathematical formulation will be given for the insertion problem. The flow on each arc becomes a binary decision variable. Specifically, π_{mthq}, ϕ_{mth} and ω_{mth} denote the flows on production, advance the clock, and idle arcs, respectively.

The first set of constraints are conservation of flow constraints at the nodes of the production network.

$$\phi_{m,t-1,h} + \omega_{m,t,h-1} + \sum_q \pi_{m,t,h-sut-tpp\cdot q,q} = \phi_{m,t,h} + \omega_{m,t,h} + \sum_q \pi_{m,t+1,h,q} \quad \forall\, m, 0 < t < Z + L(m), h$$

For the source node, $(m, 0, 0)$, supply is one unit. Thus $1 = \phi_{m,0,0} + \sum_q \pi_{m,1,0,q} \forall m$

Production batches retain their size and identity as they move through the factory. Thus, we have

$$B_{tq} = \sum_h \pi_{m,t-\theta_m,h,q} \quad \forall\, m, t, q$$

From (31), it readily follows that the earliest completion time of a batch is the maximum of the cumulative nominal lead time $\theta_m + L(m)$. This observation helps to reduce the size of the formulation by starting the time at $t = \max_{m \in M}\{\theta_m + L(m)\}$.

The constraints introduced so far do not force production to keep pace with the client's requested shipment schedule. We accomplished this by introducing an extra machine, indicated by 0, with $L(0) = 0$, $SUT_0 = 0, tpp_0 = 1$ and $\theta_0 = 0$. Since producing more than the total quantity D to be shipped is not viable, $l_{0t} = D, \forall\, t$. We set e_{0t} equal to the cumulative quantity to be shipped by time $t-1$. This matches the solid line in figure 1, offset by 1 time unit.

Before discussing the objective function we consider the eize of the constraint set. Let $\Delta = \max\{L(m) : m \in M\}$. By (14)-(17), $\Delta \geq l_{mt} - e_{mt}, \ \forall \ t$. Also let U be the largest batch size allowed. There will be approximately $|M| \ Z \ \Delta + |M| \ Z \ U$ rows (constraints) in the integer programming formulation. Since $tpp(m)$ is a positive integer, (11) implies $U \leq \Delta$, so the number of rows will be of order $|M| \ Z \ \Delta$. Note that the number of rows is approximately equal to the total number of nodes in all of the production networks. However, the number of columns (variables) is approximately $2|M| \ Z \ \Delta + |M| \ Z \ \Delta \ U + Z \ U$. Here the second term dominates, thus the number of columns will be of order $|M| \ Z \ \Delta \ U$, which is roughly equal to the total number of production arcs in the entire production network.

The cost of producing a batch of q units at time t is denoted by C_{tq}. Let K be a large constant. The objective function then is:

$$\max \ \sum_{t,q} \{qK - C_{tq}\} B_{tq}$$

Note that machine 0 does not allow us to produce more than D units. This objective function forces us to meet as much of the client's demand as possible, and in so doing to minimize setup and holding costs.

Thus, we summarize the integer programming formulation (IP1) as follows:

$$\max \ \sum_{t,q} \{qK - C_{tq}\} B_{tq} \tag{27}$$

$$\text{s.t.} \quad \phi_{m,t-1,h} + \omega_{m,t,h-1} + \sum_{q} \pi_{m,t,h-sut-tpp \cdot q,q} \tag{28}$$

$$= \phi_{m,t,h} + \omega_{m,t,h} + \sum_{q} \pi_{m,t+1,h,q} \qquad \forall \ t,h,m \in M \cup \{0\} \tag{29}$$

$$1 = \phi_{m,0,0} + \sum_{q} \pi_{m,1,0,q} \qquad \forall \ m \in M \cup \{0\} \tag{30}$$

$$B_{tq} = \sum_{h} \pi_{m,t-\theta_m,h,q} \qquad \forall \ t,h,m \in M \cup \{0\} \tag{31}$$

We used the simplex method to test how tight this formulation is for a small set of real-world data. In almost every instance, we found the optimal solution using the LP-relaxation indicating that our formulation is extremely tight.

5 Heuristics

Solving the integer programming problem formulated in section 5 is too time consuming for practical problems. Therefore, we develop a number of heuristics that are capable of handling large problems. The first

four heuristics determine the timing of a fixed number of production batches. Subsection 5.1 explains how the quantities are obtained after the timing of the batches has been determined. We search for the number of batches. First, we estimate the minimum number of batches that we need and use this estimate in our heuristics. We then iteratively increase the number of batches and run the heuristics again to see if a better solution can be found. We terminate the heuristics when no improvement has been encountered after three consecutive increases in the number of batches. Our heuristics determine only the timing of the production batches. The next subsection explains how the quantities are obtained.

5.1 Greedy Approach to Calculating Batch Sizes

Suppose that we know the number of batches $|\tau|$ and their due dates $\{t : t \in \tau\}$. We need to determine the batch sizes. Following (21)-(23) and (26), we formulate (IP2) as follows.

$$\min \sum_{t\in\tau} \left[\sum_{m\in M} S_m - g \cdot t \cdot q_t \right]$$

$$\text{(IP2)} \qquad \text{s.t.} \qquad h_{mt} \geq e_{mt} \qquad \forall\, t \in \tau \;\; \forall\, m \in M \cup \{0\} \tag{32}$$

$$h_{mt} + tpp_m \cdot q_t + sut_m \leq l_{mt} \qquad \forall\, t \in \tau \;\; \forall\, m \in M \cup \{0\} \tag{33}$$

$$h_{mt} + tpp_m \cdot q_t + sut_m \leq h_{m\sigma(t)} \qquad \forall\, t \in \tau \;\; \forall\, m \in M \cup \{0\} \tag{34}$$

where q_t is the batch quantity for the batch due on day t, and $\sigma(t)$ is the next due date in τ after t. The decision variables are h_{mt} and q_t.

A simple greedy algorithm solves (IP2). The algorithm determines batch quantities one at a time, starting with the last batch and sweeping backwards in time. If t_0 is the last batch, let $h_{\sigma(t_0)} = \infty$. At each step, we set q_t equal to the maximum quantity we can produce, namely $\min_{m\in M\cup\{0\}}\{(l_{mt} \vee h_{\sigma(t)}) - e_{mt} - sut_m)/tpp_m\}$. In Appendix C.1, we prove that this algorithm solves (IP2).

We now describe our heuristics for determining the timing of a fixed number of production batches.

5.2 Genetic Algorithm

Genetic algorithms are a class of randomized optimization techniques that are loosely modeled after the evolutionary process of natural organisms. Groups of solutions are generated iteratively using mechanisms analogous to breeding, mutation, and immigration. Genetic algorithms have received considerable attention as a means for solving a wide range of complex problems. For an introduction to the theory and application of genetic algorithms we refer the reader to Gen and Cheng (1997). A more detailed description of our implementation is given in Appendix C.2.

5.3 Simulated Annealing

Simulated Annealing can be broadly described as a randomized myopic search technique. A current solution is evaluated and compared to a randomly selected neighbor. If the cost of the neighbor is lower than that of the current solution, we update the current solution. If the cost of the new solution is higher then with some probability we accept that neighbor in order to avoid getting trapped in a local minimum. We iterate until we reach a state of equilibrium.

The probability of accepting a neighbor when its cost is higher than the current solution is determined according to a cooling schedule. A lower temperature results in a decrease of the probability that a worse solution will be accepted. Initially, we start with a temperature that is set such that the probability of accepting a solution that is worse than the current one is relatively high. For each temperature, we randomly search until we reach a state of equilibrium for that temperature. The temperature is then lowered, and we repeat this process until the temperature is sufficiently low, i.e. virtually no solutions that are worse than the current one are accepted. The best result that we found is kept.

For an introduction to the theory and application of simulated annealing we refer the interested reader to van Laarhoven and Aarts (1987).

5.4 Tabu Search

Tabu Search can be viewed as an intelligent implementation of a greedy approach. A neighborhood is defined in the following way. A particular set of batch times is a neighbor of another set of batch times if corresponding batch times from both sets differ by exactly 1 unit of time for exactly 1 batch. At each iteration, after searching through the neighborhood of the current solution, the next solution is selected. A tabu set is used to prevent the algorithm from going back to previously examined solutions for a certain number of iterations. We construct the tabu set by recording the solutions in the last k iterations. We define the next solution to be the first one we encounter in the neighborhood that is an improvement from the current solution. To avoid getting stuck in a local optimum, a diversification scheme is used to randomly explore other areas of the solution space. In our problem, we diversity by randomly shifting the batch times if the entire neighborhood does not yield improvements. We terminate the program after a fixed number of iterations.

5.5 Randomized Local Search

The Randomized Local Search heuristic is myopic in nature. For a fixed number of iterations, we randomly select a solution and perturb it until we find a local minimum. The output of the algorithm is the best of these local minima.

5.6 Single Machine Heuristic

The main IP is hard to solve for the following two reasons:

(i) The LP relaxation does not necessarily give an integer solution because, along with the network structures given for each machine, we also have side constraints (31) connecting these networks.

(ii) The number of variables and constraints is too large to solve any practical problem using integer programming algorithms.

The single machine heuristic tries to overcome these difficulties without abandoning the use of linear programming. The idea of the heuristic is the following.

Recall that in section 4, we introduced machine 0 to model the client's requested shipment schedule.

Step 1: Eliminate production arcs in the network for machine 0 that are clearly not going to yield feasible solutions. Specifically, eliminate the production arc $(0, t, h, q)$ if one of the following holds:

- (C1): a batch cannot end at time t because t is less than the total cumulative lead time, i.e. $t - \max_{m \in M}\{\theta_m + L(m)\} \ < \ 0$
- (C2): there is inadequate capacity on one or more of the machines, i.e. if for some machine $m \in M$, we have $sut_m + tpp_m \cdot q > l_{m,t-\theta_m} - e_{m,t-\theta_m}$.

Step 2: Find a shortest path from node $(0, 0, 0)$ to node $(0, z, v_0)$ in the network of machine 0. The arc lengths are given by (26). If there is no path from node $(0, 0, 0)$ to node $(0, z, v_0)$ then stop; the heuristic failed to find a feasible solution. Determine production batches by simply taking $B_{tq} = \sum_h \ \pi_{0thq}$, where π_{0thq} are the arcs on the shortest path.

Step 3: Using the logic of (24), (25), and (22), test these production batches for feasibility on each machine $m \in M$. If they are feasible on all machines then stop; a feasible solution has been obtained.

Step 4: Identify the arcs in the shortest path comoputed in step 2 that led to the infeasibility identified in *Step 3*. Delete one of these arcs from the network of machine 0, and go to *Step 2*.

Note that if a feasible solution is found the first time *Step 3* is executed, it is necessarily optimal.

5.7 Round-Off Heuristic

The order insertion problem is formulated as an integer program in the previous sections.

Instead of solving the order insertion problem as an integer program, we choose to solve progressively restricted relaxations of the IP. In this iterative procedure, the linear relaxation is solved while some batch variables are fixed to zero or one. In the first iteration, the relaxation is solved without artificially setting any variables. In a other iterations, variables are fixed according to their proximity to zero or one in the solution of the previous iteration.

In actual tests, either the problems were too large for the computer memory, or the linear relaxation returned integral solutions due to the tight formulation. Hence, the round-off heuristic was never tested.

6 Computational Results

We want to select a heuristic that performs well under a variety of circumstances, providing good feasible solutions within a reasonable amount of time. The goal is to determine whether a new order request can be accepted while the client is still on the phone, so a "reasonable" amount of time is something on the order of a few minutes. We therefore created three versions of the Genetic Algorithm (GA), Simulated Annealing (SA), Tabu, and Randomized Local Search heuristics. Each heuristic was limited to 30 seconds, 1 minute, and 5 minutes. This gave us twelve heuristics to compare, in addition to the Single Machine heuristic. We tested the heuristics on a wide range of realistic problems similar to the data obtained from the company.

6.1 Problem Generation

We designed the test problems so that each has a feasible solution. The following are the main steps in generating a test problem.

(i) Generate a bill of material (BOM) tree whose depth is parametrically specified by the user.

(ii) Batch order quantities are generated randomly.

(iii) For each machine nominal lead times $L(m)$ are set equal to NLT (nominal lead time multiplier) times the processing time of the longest operation on m.

(iv) Batch due dates are computed ensuring that they are larger than the length of critical path.

(v) For each production batch, a "new operation" is scheduled on each machine.

(vi) Shipment quantities are generated so that there are a given number of shipments.

(vii) Shipment due dates are chosen randomly but with the constraint that at every time period cumulative batch quantities completed have to be larger than or equal to cumulative shipments.

(viii) Additional operations are added to the schedule of each machine until a pre-specified level of utilization is reached.

(ix) The "new operations" created in Step 5 are removed from the schedule, of each machine, to create the production plan.

6.2 Experimental Design

We performed three sets of experiments. In the first set we compared the heuristics to the lower bounds obtained with the Round-Off heuristic. The size of the problems was limited by the amount of memory required by AMPL. Our second set of experiments tested the heuristics against one another on a collection of larger problems. Problems for the first two experiments were generated randomly. Our third set of experiments was based on actual data provided by the manufacturer.

In the first two sets of experiments, we identified a number of parameters used in problem generation. Those parameters include the number of time periods, the number of machines, the BOM-trees, the number of shipments and the number of batches in the feasilbe schedule we created. These parameters determined the probability distributions used to generate the test problems. We fixed a base case that represented a realistic setting of the parameter values, and for each parameter we chose a set of additional values to test.

One at a time, we varied each parameter over its range of possible values, leaving the other parameters at their base case settings. For each parameter value, we generated a common set of problems that were used to test the heuristics. We computed the average cost of each heuristic over the set of problems, as well as the fraction of problems for which that heuristic found a feasible solution.

All computational tests were performed on a SPARC Ultra-1 Sun station.

6.3 Results for the First Experiment: Small-Scale Problems

We generated 6 random problems for a base case and 3 random problems for each of 15 variations of this base case. All 51 problems (with time horizons of approximately 70 periods) were solved by our Round-Off heuristic to optimality. Surprisingly, the solution to the LP-relaxation was binary for every one of the 51 problems. This very strongly indicates the quality of the formulation in terms of tightness. Table 1 gives the

average ratio of cost to optimal cost for each of the other heuristics. (The results in Table 1 are averaged over 49 problems. For two of the problems the optimal cost was 0; in these cases the heuristics were optimal or close to it.) Table 1 indicates that for these problems the better heuristics are close to optimal and that

Heuristic	**Seconds**	**Avg. Cost/Optimal**
GA	30	1.04
Random	30	1.07
SA	30	1.10
Tabu	30	1.39
GA	60	1.04
Random	60	1.06
SA	60	1.11
Tabu	60	1.39
GA	300	1.04
Random	300	1.06
SA	300	1.13
Tabu	300	1.39

Table 1: Cost Comparison; Small-Scale Problems

increasing the time limits had little effect.

6.4 Results for the Second Experiment: Full-Scale Problems

We used thirty-five parameter settings to generate sample problems. The parameters were chosen to span a wide range of potential situations. Each parameter setting was replicated 20 times, for a total of 700 problems. In order to get a general picture of how the heuristics compare, we averaged the performance measures over all of the parameter settings.

We first looked at how well the heuristics performed in terms of finding feasible solutions to the 700 sample problems. For each heuristic, we calculated the fraction of problems where the heuristic found a feasible solution (Table 2). As expected, within each heuristic the fraction of feasible solutions increases with the time spent on the problem. The jumps from 30 to 60 to 300 seconds seem to be both economically and statistically significant. Practically speaking, one would probably want a version between 60 and 300 seconds.

Heuristic	**sec.**	**% Feasible**
GA	30	83.6
Random	30	87.4
SA	30	89.0
Tabu	30	74.3
GA	60	96.1
Random	60	89.9
SA	60	95.3
Tabu	60	72.6
GA	300	99.7
Random	300	95.7
SA	300	98.9
Tabu	300	85.3
Single Machine		60.9

Table 2: % of Feasible Solutions Found

Our implementation of the Tabu heuristic seems not to be as good at finding feasible solutions as the other heuristics. The Single Machine heuristic is also not as good at finding feasible solutions, although this is largely due to a CPU constraint. In 82% of the problems where the Single Machine heuristic did not find a feasible solution, it was due to lack of memory for AMPL. This may improve with re-coding.

We next compared the heuristics on the basis of average relative cost: the heuristic's cost divided by the lowest-cost solution from any of the 13 algorithms (Table 3). For each time limit, we averaged the relative cost over problems where all four of the heuristics found a feasible solution. The Single Machine heuristic was compared to the the 300 second group. We compared heuristics on the basis of relative cost to avoid giving too much weight to problem instances that have high costs. The Randomized Local Search heuristic was not good at finding low-cost solutions relative to the other heuristics. The Single Machine heuristic was only slightly better than the 300-second Simulated Annealing and Genetic Algorithm heuristics, yet it was much worse at finding feasible solutions.

We also investigated the average time to completion for each of the heuristics (Table 4). For a particular heuristic, the average was taken over all problems where the heuristic found a feasible solution. Since the algorithms are random in nature, it was difficult to control the exact amount of time spent (particularly for our implementation of the Simulated Annealing algorithm). However, on average, the times were close to

Heuristic	Sec.	Avg. Cost/Best	St. Dev.	Median	Min.	Max.	# of Prob.
GA	30	1.32	0.76	1.20	1.00	9.59	485
Random	30	1.49	0.69	1.41	1.00	9.21	485
SA	30	1.29	0.55	1.20	1.00	6.83	485
Tabu	30	1.49	0.91	1.32	1.00	13.30	485
GA	60	1.23	0.38	1.17	1.00	4.82	494
Random	60	1.41	0.65	1.34	1.00	8.76	494
SA	60	1.18	0.32	1.12	1.00	4.91	494
Tabu	60	1.36	0.68	1.20	1.00	9.28	494
GA	300	1.10	0.16	1.07	1.00	3.21	589
Random	300	1.27	0.47	1.22	1.00	6.60	589
SA	300	1.06	0.11	1.00	1.00	2.91	589
Tabu	300	1.20	0.34	1.08	1.00	3.95	589
Heuristic	**Sec.**	**Avg. Cost/Best**	**St. Dev.**	**Median**	**Min.**	**Max.**	**# of Prob.**
GA	300	1.10	0.18	1.06	1.00	3.21	388
Random	300	1.27	0.55	1.19	1.00	6.60	388
SA	300	1.07	0.13	1.01	1.00	2.91	388
Tabu	300	1.23	0.37	1.08	1.00	3.95	388
Single Machine		1.05	0.27	1.00	1.00	4.81	388

Table 3: Cost Comparison

Heuristic	Sec.	Avg. Time	St. Dev.	Median	Min.	Max.	# of Prob.
GA	30	28.8	4.4	30.1	7.3	30.5	585
Random	30	24.7	8.2	30.0	1.5	30.4	612
SA	30	29.7	9.0	32.4	2.2	46.1	623
Tabu	30	25.6	7.0	30.0	2.9	31.8	520
GA	60	52.1	13.2	60.1	7.8	60.7	673
Random	60	48.1	16.7	60.0	3.1	60.4	629
SA	60	53.5	18.5	60.6	4.2	91.3	667
Tabu	60	53.6	12.1	60.0	8.0	62.4	508
GA	300	192.3	75.7	186.0	22.5	300.6	698
Random	300	237.5	80.9	293.8	15.7	300.4	670
SA	300	201.8	103.0	191.0	7.4	440.1	692
Tabu	300	256.0	68.0	300.0	29.8	301.7	597
Single Machine		329.8	300.4	311.5	2.0	1242.0	426

Table 4: CPU-Time Comparison

the specified limits. The Single Machine heuristic, whose completion time was not restricted, had an average time comparable to the 300-second versions of the other heuristics, but with a much higher variability. The maximum was over 20 minutes.

On the basis of the these three tables, the most interesting comparison seems to be between the Simulated Annealing and Genetic Algorithm heuristics. Table 5 show the fraction of problems where SA found a lower-cost solution than GA (considering only problems where both found a feasible solution). Table 6 shows

	GA30	GA60	GA300
SA30	0.48	0.36	0.12
SA60	0.63	0.55	0.24
SA300	0.82	0.81	0.57

Table 5: Fraction in which SA Beat GA in Cost

the fraction of problems where GA found a lower-cost solution than SA. The difference between the sum of the two tables and 1 is the fraction of problems where the two heuristics tied. The Simulated Annealing heuristics slightly outperformed the Genetic Algorithm overall. The fraction of problems where SA finds a lower cost then GA increases as the time limit increases (although this may be due to the imprecise time

	SA30	SA60	SA300
GA30	0.42	0.26	0.08
GA60	0.54	0.35	0.10
GA300	0.77	0.64	0.28

Table 6: Fraction in which GA Beat SA in Cost

limits for SA). However the difference in average relative costs remains about the same with changing time limits (Table 3).

We also examined the effects of varying particular parameters in the problem generation scheme, such as the number of machines, depth of the BOM, bounds for batch sizes, set up times, processing times, NLT, set up costs, and utilization percentages. For each parameter setting we generated 20 independent problem instances. We will describe how changes in these parameters affect costs and feasibility for the Simulated Annealing and Genetic Algorithm heuristics. As a reminder, costs are composed of finished good inventory holding costs and set up costs.

As the NLT increases, both costs and percent feasibility (see Figure 7) increase. Larger values of NLT

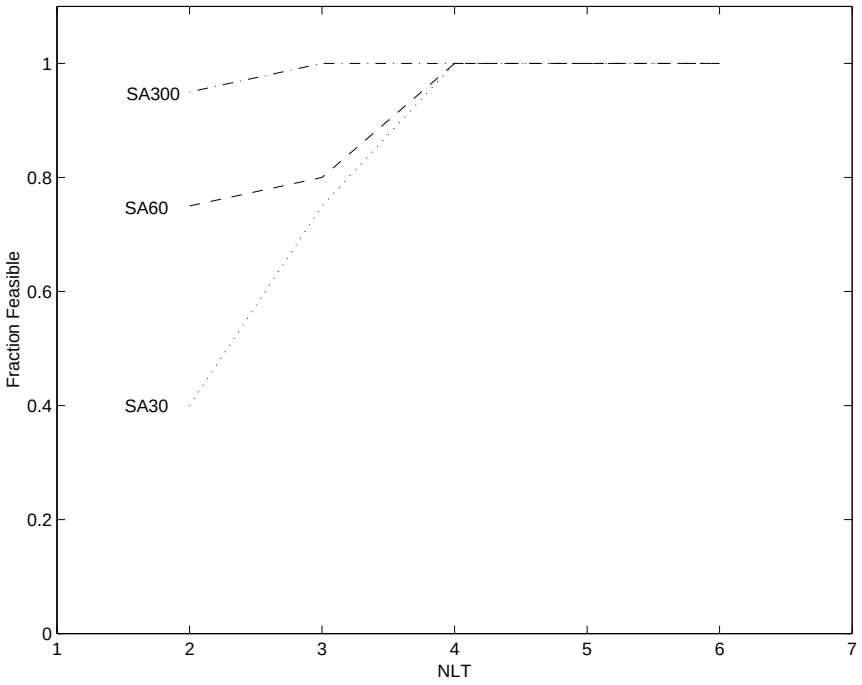


Figure 7: Fraction of Feasible Solutions Found vs. NLT

lead to longer time horizons, increasing holding costs. On the other hand, larger values of NLT result in

larger time windows on each machine where operations can be inserted. Hence, detection of feasibility is improved.

As the number of time periods increases, both holding costs and total costs increase. Detection of feasibility initially deteriorates (see Figure 8). The horizontal axis is the difference between the time horizon

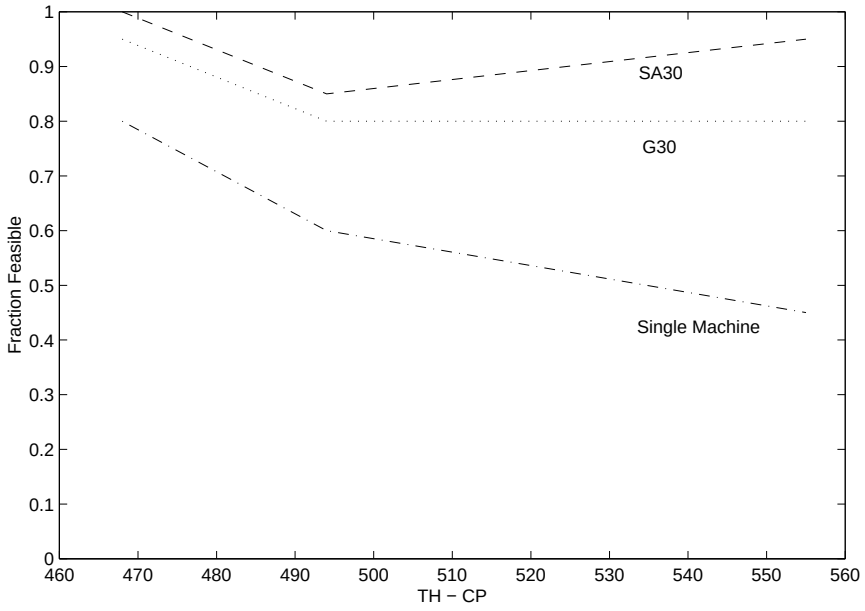


Figure 8: Fraction of Feasible Solutions Found vs. TH-CP

(TH) and the critical path (CP:=max$\{L(m) + \theta(m) : m \in M\}$). Longer time horizons allow more flexibility in scheduling batches. On the other hand, longer horizons can cause our heuristics to time out before finding a feasible solution. The Single Machine Heuristic does very poorly with long time horizons, but this is related to the memory required by AMPL.

We also varied other parameters and examined the results in terms of cost and feasibility. In these cases, we did not encounter any surprises. In summary, costs increased with the number of machines, number of shipments, and utilization percentages. On the other hand, detection of feasibility improved with fewer machines, fewer shipments, and smaller utilization percentages.

6.5 Results for the Third Experiment: Real-World Data

In order to generate sample problems, we took an existing feasible schedule that included 421 products and 43 workcenters. The length of the schedule was 1400 time periods. Since the factory runs three 8-hour shifts 5 days/week, the schedule corresponded to about three months. Setup times were rounded to 1, 2, or 3 hours; processing time per part (TPP) was rounded to 1, 2, or 3 hours/100 pieces. We formed seven sample

insertion problems by removing multi-batch quotes for seven different products.

The results of the seven problems are given in Table 7. In general, all heuristics were able to find feasible

Problem #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**Overall**
# of Shipments	3	8	4	2	3	7	2	**Average**
# of Workcenters	7	2	2	2	3	3	5	
	$\frac{Cost}{Best}$	$\frac{Cost}{Best}$	$\frac{Cost}{Best}$	$\frac{Cost}{Best}$	$\frac{Cost}{Best}$	$\frac{Cost}{Best}$	$\frac{Cost}{Best}$	$\frac{Cost}{Best}$
GA30	1.20	1.40	2.14	1.00	1.08	1.66	1.07	1.36
Random30	1.29	1.55	1.60	1.00	1.28	1.78	1.23	1.39
SA30	1.19	1.33	1.55	1.00	1.18	1.75	1.13	1.31
Tabu30	1.26	1.34	2.16	2.22	1.17	1.28	1.75	1.60
GA60	1.20	1.34	2.14	1.00	1.08	1.49	1.07	1.33
Random60	1.07	1.25	2.65	1.00	1.14	1.42	1.19	1.39
SA60	1.00	1.16	1.32	1.00	1.11	1.73	1.01	1.19
Tabu60	1.00	1.34	2.22	1.99	1.04	1.52	1.57	1.53
GA300	1.00	1.05	1.76	1.00	1.06	1.08	1.09	1.15
Random300	1.05	1.27	1.81	1.00	1.05	1.33	1.05	1.22
SA300	1.19	1.00	1.50	1.00	1.00	1.46	1.00	1.17
Tabu300	1.00	2.35	1.88	1.00	1.17	1.90	1.00	1.47
Single Machine	1.00		1.00	1.00	1.00	1.00	1.00	1.00
Best Cost	2112	3740	7000	700	5176	7977	7700	
Single Machine Feasible?	yes	no	yes	yes	yes	yes	yes	
Single Machine Provably Optimal?	yes	no	no	yes	yes	no	no	
Single Machine Time	18		385	12	85	1112	796	

Table 7: Industrial Problems

solutions, although the costs were volatile. Quotations with a greater numbers of shipments seemed to be more difficult. There is more randomness in performance here than in the other analysis because we are not averaging over many problem instances.

7 Conclusions

We have modelled a job insertion problem, and proven it to be NP-hard. We have developed and tested a number of heuristics, using both randomly-generated and real-world problem instances. Several of the heuristics look very promising, in particular, the genetic algorithm, simulated annealing, and single machine heuristics.

The biggest limitation for our computational work is that we treat time as a discrete variable measured in hours. All operation start times and processing times are an integral number of hours. This is not adequate for a realistic industrial setting unless time periods are shortenend.

The continuous relaxation of our integer programming formulation was solved for 60 randomly generated problem instances. In every case the solution was integral. The formulation seems to be remarkably tight.

Areas for future research include developing approximation algorithms, a theoretical study of properties of the polytopes, and improving the heuristics to handle a greater number of time periods or to work in continuous time.

8 Acknowledgements

We are greatly indebted to Prof. Peter L. Jackson, who worked with the parts manufacturer and provided information, comments and insights, and assistance with data for the industrial problems.

A NP-Hardness Proof

In this appendix we prove the following theorem.

Theorem 2 *The Order Insertion problem is NP-hard.*

Proof. The proof is by reduction from the (unweighted) Vertex Packing Problem, see Nemhauser and Wolsey (1988). We use the optimization version of the Vertex Packing Problem, which can be stated as follows: given a graph G with vertex set V and edge set A, find a subset V^* of V with maximal cardinality, such that no edge in A is adjacent to two different vertices of V^*.

Let v be the number of vertices in V, let a be the number of edges in A, and let V^* be an (unknown) optimal vertex packing. Given an instance on the Vertex Packing Problem, we will construct an instance of the Order Insertion Problem in which the Vertex Packing Problem has an optimal cost of $2a + 2v - |V^*|$.

We construct our instance of the Order Insertion Problem as follows. The setup times $s(m)$ are all equal to 0 and the times per part $tpp(m)$ are all equal to 1. Thus all of the run lengths are equal to the batch sizes. We set the holding costs equal to 0 and the setup cost equal to 1, so that the total cost is the number of batches. There is a machine m, $1 \leq m \leq v$, in the Order Insertion Problem for each vertex m in G. In addition, there are two special machines, machine $v+1$ (called the "Timer") and machine $v+2$ (called the "Blocker"). The available capacity on the Blocker is $3a + 2v$, which is equal to the total capacity required to fill the order. Consequently, we have

Claim 1 *In every feasible solution to the Order Insertion Problem, all $3a + 2v$ units of available capacity on the Blocker are utilized.*

We construct the capacity graph of the Order Insertion Problem by dividing the time horizon $\{\delta : 0 \leq \delta \leq H\}$ into three sections. The first section, called the "Start Section," consists of the time interval $\delta \in [0, 2]$. For each machine m the functions $\ell(\delta)$ and $e(\delta)$, $\delta \in [0, 2]$ are referred to as the "Start Structure" for m. In Figure 9, the upper curve is $\ell(\delta)$ and the lower curve is $e(\delta)$. The horizontal "calendar time" axis and the vertical "capacity" axis are omitted.

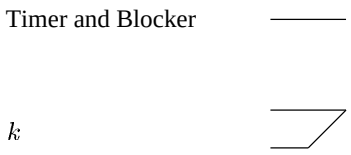


Figure 9: Start Structure

Consider all bars in the capacity graph for machine m, whose time coordinates fall in $[0, \delta]$ for some $\delta > 0$. Let $\zeta(m, \delta)$ be the vertical coordinate of the top end of the last of these bars. We define the "unused capacity" on machine m at time δ to be $\ell(\delta) - \max\{u(\delta), \zeta(m, \delta)\}$. For machines m, $m \leq v$, we have $e(\delta) = 0$ and $\ell(\delta) = \min\{\delta, 1\}$ for $0 \leq \delta \leq 2$. Because machines $v+1$ and $v+2$ have $e(\delta) = \ell(\delta) = 0$ for $0 \leq \delta \leq 2$, no batches can be inserted in this section. Consequently the Start Structure creates a single unit of unused capacity on each machine m, $m \leq v$. In the second section of the time horizon, many of these machines will lose that unit of unused capacity. The machines $m \leq v$ that still have a unit of unused capacity at the end of the second section correspond to the nodes $m \in V^*$ that are in the node packing.

The "Edge Section", the second section of the time horizon, consists of a sequence of blocks of time called "Edge Blocks", one for each edge $(m, k) \in A$ of the graph G. Suppose that the e-th Edge Block corresponds to edge (m, k). At the end of the Edge Block corresponding to (m, k), either machine m or machine k (or both) will lack a unit of available capacity and, consequently, will not be a candidate to be

in the node packing. The Edge Block extends from time θ to time $\theta + 16$, where $\theta = 2 + 16(e-1)$. The functions $\ell(\delta)$ and $e(\delta)$, $\delta \in [\theta, \theta{+}16]$ are referred to as the "Edge Structure" for (m,k). For each machine, the upper curve in Figure 10 is $\ell(\delta)$, the lower curve is $e(\delta)$, and the horizontal and vertical axes are omitted. The vertical bars in the figure are possible locations for the bars that correspond to production batches. Thus, for the Timer (machine $m = v + 1$), the functions $\ell(\delta)$ and $e(\delta)$ are equal to each other at times $\delta \in \{\theta, \theta+2, \theta+6, \theta+10, \theta+14, \theta+16\}$.

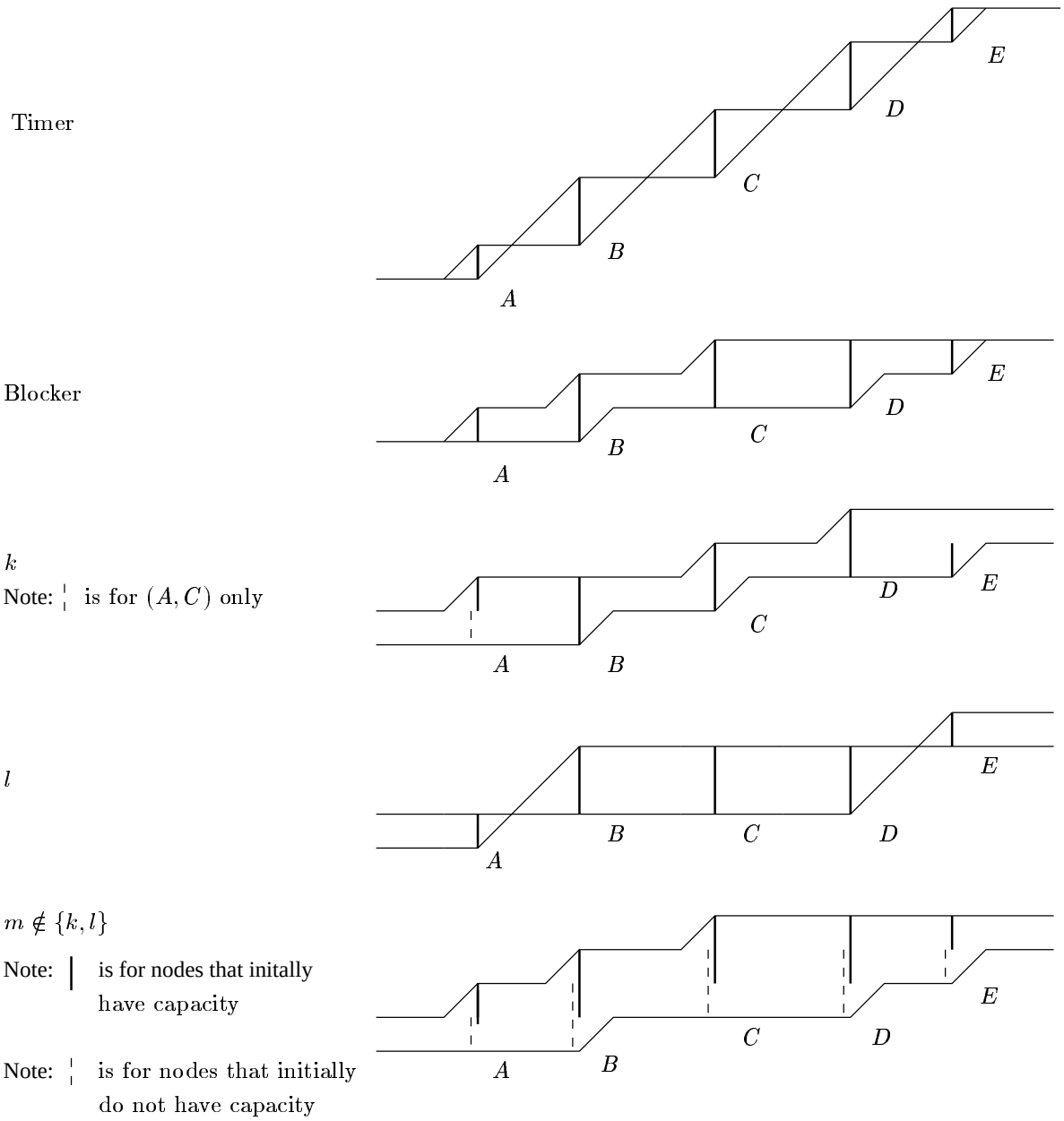


Figure 10: The Edge Structure for Edge $\{m,k\}$

We are interested in solutions to the Order Insertion Problem that have certain properties, the first of which is listed below.

Property 1 *The number of batches in any Edge Block is at most two.*

The labels A, B, C, D, E in figure 10 correspond to times θ+1, θ+4, θ+8, θ+12, θ+15.

Claim 2 *In all feasible solutions, at least two batches are scheduled in each Edge Block.*

If Property 1 holds then exactly two batches are scheduled. The batches have one of the following pairs of due dates: (A,D), (B,E) and (C,E). The batch that has due date A or E is of size 1, and the other batch is of size 2.

Proof. By Claim 1, a feasible solution will use all of the capacity that exists on the Blocker in every Edge Block. Since $e(\delta)$ and $\ell(\delta)$ are equal to each other at times θ and $\theta + 16$, the 3 units of capacity that the Blocker has between time θ and time $\theta + 16$ must be used during this time interval. The Edge Structure for the Blocker dictates that at least 2 production batches will be used (exactly 2 if Property 1 holds). The Edge Structure for machine k in Figure 10 implies that the batch sizes will be 1 and 2, and that the batch of size 1 must be produced either at or before time A or at or after time E. The Edge Structure for Timer (machine $v + 1$) constrains all of the batches to be produced at one of the points in time labeled A, B, C, D, E. Machine $v + 2$ (Blocker) implies that the only admissible pairs of batches are (A,D), (B,E) and (C,E). This proves Claim 2. ■

A "Low-Bar Solution" is a solution to the Order Insertion Problem that has the following property: If b is a production batch, then for every machine m, the bar that corresponds to machine m and batch b cannot be lowered without making the solution infeasible. Since lowering a bar without making the solution infeasible has no impact on costs, we have

Claim 3 *Given any feasible solution that satisfies Property 1, there is a Low-Bar Solution that is feasible, has the same cost, and satisfies Property 1.*

Suppose we were to modify our instance of the Order Insertion Problem by retaining the time interval $[0, \delta]$ and discarding the rest of the time horizon. A "δ-feasible solution" is a feasible solution to this restricted version of the Order Insertion problem which satisfies Property 1, is a Low-Bar solution, and fully utilizes the capacity that is available on the Blocker.

Claim 4 *Consider the Edge Block corresponding to edge (m,k) and starting at time θ.*

(i) Let $j \notin \{k,m\}$, $1 \leq j \leq v$. In all $(\theta+16)$ - feasible solutions, j has a unit of unused capacity at time θ if and only if j has a unit of unused capacity at time $\theta+16$.

(ii) There is a $(\theta+16)$ - feasible solution that uses due dates (A,D) in the Edge Block if and only if machine k has a unit of unused capacity at time θ. If due dates (A,D) are selected then machine k will have a unit of unused capacity at time $\theta+16$, but machine m will not.

(iii) There is a $(\theta+16)$ - feasible solution that uses due dates (B,E) in the Edge Block if and only if machine m has a unit of unused capacity at time θ. If due dates (B,E) are selected then machine m will have a unit of unused capacity at time $\theta+16$, but machine k will not.

(iv) There is always a $(\theta+16)$ - feasible solution that uses due dates (C,E) in the Edge Block. If (C,E) are selected, then neither machine k nor machine m will have any unused capacity at time $\theta+16$.

The proof of Claim 4 is omitted. Induction based on Claims 2 and 4 implies

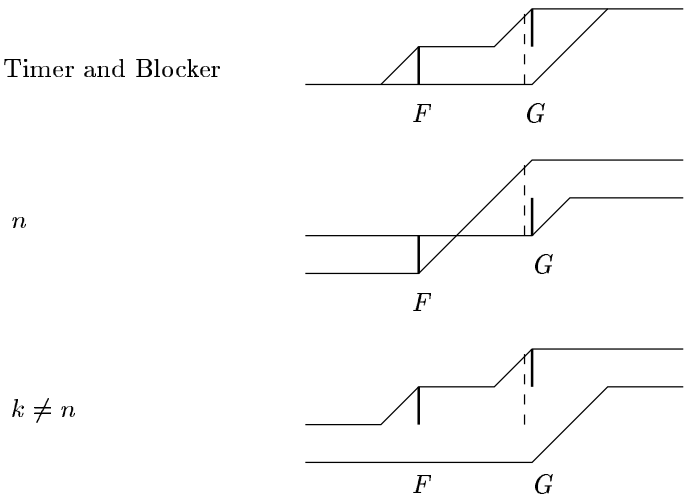
Claim 5 *Let the Edge Section end at time δ'. Let S^* be the set of δ'-feasible solutions. Let $V \subset N$ be a set of vertices, and suppose that G has at least one edge. Then V is a feasible node packing in G if and only if there is a solution in S^* for which the machines in V have a unit of unused capacity at time δ'.*

Note that all solutions in S have the same cost in the first two sections of the time horizon. The third and final section of the time horizon is the "Vertex Section". The Vertex Section counts the vertices which have a unit of unused capacity at the end of the Edge Section, by incorporating $|V|$ into the cost. The Vertex Section consists of a "Vertex Block" for each $m, 1 \leq m \leq v$. It extends from time η to time $\eta+6$, for some value of η. The times $\eta+2$ and $\eta+5$ are labeled F and G, respectively. The Vertex Structure for m consists of the functions $\ell(\delta)$, $e(\delta)$ for $\delta \in [\eta, \eta+6]$. It is illustrated in Figure 11.

We are interested in $(\eta+5)$ - feasible solutions that satisfy that satisfy the following property.

Property 2 *The number of batches in the Vertex Block for m is one if m has a unit of unused capacity at time η. Otherwise the number of batches is two. Let $k \neq m$ at time η. Then k has a unit of unused capacity at time η if and only if k has a unit of unused capacity at time $\eta+6$.*

Claim 6 *An $(\eta+5)$ - feasible solution that satisfies Property 2 exists. Suppose that two $(\eta+5)$ - feasible solutions are identical in the Start Section and in the Edge Section. Then the one that satisfies Property 2 has the lower cost, or both have the same cost.*



Note: | is used if node n has available capacity at the end of the structure.

¦ is used otherwise.

Note: | and ¦ are translated down one unit if node k does not have a unit of available capacity at the end of the structure.

Figure 11: Node Structure

Proof. By Claim 1, a total of two units is produced in each vertex structure. The Vertex Structure for m implies that at least as many setups as are required by Property 2 are performed by any $(\eta + 5)$ - feasible solution. By induction on m, an $(\eta + 5)$ - feasible solution that satisfies Property 2 exists (see figure 11). When the m-th Node Block begins, the $(\eta + 5)$ - feasible solution that satisfies Property 2 has at least as much unused capacity on each of the machines as the other $(\eta + 5)$ - feasible solution.

If V^* is a feasible node packing, we have shown that there is a feasible solution to the Order Insertion Problem in which has a cost of $2a + 2v - |V^*|$. Suppose that we are given a feasible solution to the Order Insertion Problem, and that V' is the set of machines that have a unit of unused capacity at the end of the Edge Section. Then V' is a feasible node packing, and the cost of the solution is at least $2a + 2v - |V'|$. ■

B Proof of Theorem 1

Suppose that $d(k) \leq \delta < d(k+1)$. Then

$$\mathcal{L}(\delta) = T(k) \vee \max_{k_* > k}\{T(k_*) - (d(k_*) - \delta)\} = \max_{k_* \geq k}\{T(k_*) - [d(k_*) \vee \delta] + \delta\}. \tag{35}$$

Similarly, suppose that $d(j-1) < \delta \le d(j)$. Then

$$\mathcal{E}(\delta - L) = \min_{j_* < j}\{\delta - d(j_*) + T(j_* - 1)\} \wedge T(j-1) = \min_{j_* \le j}\{\delta - [d(j_*) \wedge \delta] + T(j_* - 1)\}. \tag{36}$$

To prove Theorem 1, we need the following Lemmata.

Lemma 5 *The following hold:*

(i) $d(k) \le \delta < d(k+1) \quad \Longrightarrow \quad l(\delta) \le \delta - T(k)$

(ii) $d(j-1) < \delta \le d(j) \quad \Longrightarrow \quad e(\delta) \ge \delta - L - T(j-1)$

Proof.

$$\begin{aligned} \mathcal{L}(\delta) &\overset{(35)}{\ge} T(k) \quad &\Longrightarrow \quad & l(\delta) \le \delta - T(k) \\ \mathcal{E}(\delta - L) &\overset{(36)}{\le} T(j-1) \quad &\Longrightarrow \quad & e(\delta) \ge \delta - L - T(j-1) \end{aligned}$$

■

Let $T'(k) := \sum_{k_* \le k} p'(k_*)$.

Lemma 6 *Suppose that* $\{d'(k), p'(k), c'(k) : 1 \le k \le J'\}$ *is a feasible quotation. Then* $T'(k) \le p'(k) + c'(k)$. *If* $j \le k$ *then* $T'(k) - T'(j-1) \le p'(k) + c'(k) - c'(j)$.

Proof. The result is obtained by summing (23) with successive values of k. ■

Proof of Theorem 1. We combine the operations $\{d(k), p(k) : 1 \le k \le J\}$ in the current workload and the operations $\{d'(k'), p'(k') : 1 \le k' \le J'\}$ in the feasible quotation into a single set, and re-index them as $\{d''(k''), p''(k'') : 1 \le k'' \le J''\}$ where $J'' = J + J'$ and $d''(k'') \le d''(k''+1)$. For convenience only, we assume that $d''(k'') < d''(k''+1)$.

We define the functions $k(k'')$, $k'(k'')$, $j(j'')$, and $j'(j'')$ by the following:

$$\begin{aligned} d(k(k'')) &\le d''(k'') < d(k(k'')+1) \\ d'(k'(k'')) &\le d''(k'') < d'(k'(k'')-1) \\ d(j(j'')) &\ge d''(j'') > d(j(j'')-1) \\ d'(j'(j'')) &\ge d''(j'') > d'(j'(j'')-1) \end{aligned}$$

We must show that (9) holds if the workload on machine m is

$$\{d''(k''), p''(k'') : 1 \le k'' \le J''\}. \tag{37}$$

and if

$$d''(j'') - L \quad \leq \quad d''(k''). \tag{38}$$

Let $k := k(k'')$, $k' := k'(k'')$, $j := j(j'')$, and $j' := j'(j'')$. Then

$$T(k) + T'(k') \quad = \quad T''(k) \tag{39}$$

$$T(j-1) + T'(j'-1) \quad = \quad T''(j''-1) \tag{40}$$

$$\tag{41}$$

Let $B := T''(k'') - T''(j''-1) + d''(j'') - L-)$. Then (9) is equivalent to $0 \geq B$. If $k'' < j''$ then the sum of the first two terms of B is less than or equal to zero. By our assumption, the sum of the last three terms is also less than or equal to zero, and (9) holds. Assume that $k'' = j''$ and $d''(k'') = d(k)$. Then (9) is a consequence of (11). Thus we can assume that $k'' \geq j''$, and that if $k'' = j''$ then $d''(k'') = d'(k')$. Thus $k' \geq j'$.

By Lemma 6, (21)-(23), the monotonicity of $l(\cdot)$ and $e(\cdot)$, and Lemma 5,

$$\begin{aligned} T'(k') - T'(j'-1) &\leq p'(k') + c'(k') - c'(j') \\ &\leq l(d'(k')) - e(d'(j')) \\ &\leq l(d''(k'')) - e(d''(j'')) \\ &\leq [d''(k'') - T(k)] - [d''(j'') - L - T(j-1)]. \end{aligned} \tag{42}$$

Consequently by (39) and (42),

$$\begin{aligned} B &= [T(k) - T(j-1)] + [T'(k') - T'(j'-1)] + [d''(j'') - L - d''(k'')] \\ &\leq [T(k) - T(j-1)] + [d''(k'') - T(k) - d''(j'') + L + T(j-1)] + [d''(j'') - L - d''(k'')] \\ &= 0 \end{aligned}$$

This proves the first assertion.

To prove the second assertion, we assume that the current workload $\{d(k), p(k) : 1 \leq k \leq J\}$ admits a feasible schedule, and that that we are given a quotation $\{d'(k'), p'(k') : 1 \leq k' \leq J'\}$. We assume that the workload given by (37) admits a feasible schedule. We need to show that the quotation is feasible. Let $c'(\cdot)$ be given by (24)-(25). Then clearly (21) and (23) hold. It suffices to show that $c'(k') + p'(k') \leq l(d'(k'))$ $\forall\, k'$.

A simple induction proof based on (24)-(25) implies that there is a $j' \leq k'$ such that

$$c'(k') + p'(k') = e(d'(j')) + \sum_{k'_* = j\prime}^{k\prime} p'(k'_*) = e(d'(j')) + T'(k') - T'(j'-1).$$

Thus, it suffices to show that

$$\begin{aligned} 0 &\geq e(d'(j')) - l(d'(k')) + T'(k') - T'(j'-1) \\ &= d'(j') - L - \mathcal{E}(d'(j') - L) - d'(k') + \mathcal{L}(d'(k')) + T'(k') - T'(j'-1). \end{aligned}$$

We define k, k'', j and j'' by

$$d(k) < d'(k') = d''(k'') < d(k+1) \tag{43}$$

$$d(j-1) < d'(j') = d''(j'') < d(j). \tag{44}$$

Recall that $j' \leq k'$, so $j'' \leq k''$. By (35) and (36), it suffices to show that

$$\begin{aligned} 0 &\geq d'(j') - L - d'(k') + T'(k') - T'(j'-1) - \mathcal{E}(d'(j') - L) + \mathcal{L}(d'(k')) \\ &= d'(j') - L - d'(k') + T'(k') - T'(j'-1) \\ &\quad - \min_{j_* \leq j} \{d'(j') - [d(j_*) \wedge d'(j')] + T(j_*-1)\} \\ &\quad + \max_{k_* \geq k} \{T(k_*) - [d(k_*) \vee d'(k')] + d'(k')\} \end{aligned} \tag{45}$$

Consequently, it suffices to show that $\forall\, k_* \geq k \;\; \forall\, j_* \leq j$,

$$\begin{aligned} 0 &\geq d'(j') - L - d'(k') + T'(k') - T'(j'-1) - d'(j') + [d(j_*) \wedge d'(j')] \\ &\quad - T(j_*-1) + T(k_*) - [d(k_*) \vee d'(k')] + d'(k') \\ &= [d(j_*) \wedge d'(j')] - L - [d(k_*) \vee d'(k')] + T'(k') - T'(j'-1) + T(k_*) - T(j_*-1). \end{aligned} \tag{46}$$

We define k'_*, k''_*, j'_* and j''_* by

$$d'(k'_*) < d(k_*) = d''(k''_*) < d'(k'_*+1) \tag{47}$$

$$d'(j'_*-1) < d(j_*) = d''(j''_*) < d'(j'_*) \tag{48}$$

We claim that

(i) If $k_* > k$ then $k''_* > k''$ and $k'_* \geq k'$

(ii) If $k_* = k$ then $k''_* < k''$ and $k'_* < k'$

(iii) If $j_* < j$ then $j''_* < j''$ and $j'_* \leq j'$

(iv) If $j_* = j$ then $j''_* > j''$ and $j'_* > j'$.

If $k_* > k$ then $k_* \geq k+1$. By (43)-(44) and (47)-(48), $d'(k'_*+1) > d''(k''_*) = d(k_*) \geq d(k+1) > d'(k') = d''(k'')$, so (i) holds. If $k_* = k$ then by (43)-(44) and (47)-(48), $d''(k'') = d'(k') > d(k) = d(k_*) = d''(k''_*) > d'(k'_*)$. Hence, $k''_* < k''$, $d(k_*) < d'(k')$. But (47) implies that $d(k_*) > d'(k'_*)$, so $k'_* < k'$. This establishes (ii). The proofs of (iii) and (iv) are similar.

A consequence of (43)-(44), (47)-(48) and this claim is the following:

$$\begin{aligned} T''(k''_* \vee k'') &= T'(k'_* \vee k') + T(k_*) \\ T''((j''_* \wedge j'') - 1) &= T'((j'_* \wedge j') - 1) + T(j_* - 1). \end{aligned}$$

Note that Lemma 2 holds for (37). Since $j'' \leq k''$, we can apply Lemma 2 for $k''_* \vee k''$ and $j''_* \wedge j''$. Thus,

$$\begin{aligned} 0 &\geq d''(j''_* \wedge j'') - L - d''(k''_* \vee k'') + T''(k''_* \vee k'') - T''(j''_* \wedge j'' - 1) \\ &= [d(j_*) \wedge d'(j')] - L - [d(k_*) \vee d'(k')] + T'(k'_* \vee k') - T'((j'_* \wedge j') - 1) + T(k_*) - T(j_* - 1) \end{aligned}$$

By the monotonicity of $T'(\cdot)$, (46) holds. ■

C Descriptions of Heuristics

C.1 Greedy Approach to Calculate Batch Sizes

Proposition 1 *The greedy algorithm described in section 6.1 solves (IP2) exactly.*

Proof. Suppose that the number of batches is k and the given batch times are $t_1, \ldots, t_k$. Let $q'_1, \ldots, q'_k$ be an optimal solution found by solving (IP2). Let $q_1, \ldots, q_k$ be the solution found by the greedy algorithm. Let i be the largest index such that $q'_i \neq q_i$.

Case 1: $q'_i < q_i$.

By definition of the greedy algorithm, $q_1, \ldots, q_k$ are obviously feasible for (IP2). If $q'_i < q_i$ and i is the largest such index, then we must have produced the same amount of products after period t_i in both the LP solution and the greedy solution. Since the LP solution is optimal, then $q'_1 + \ldots + q'_k \geq q_1 + \ldots + q_k$. Hence, $q'_1 + \ldots q'_i \geq q_1 + \ldots + q_i$. If $q'_i < q_i$, then $q'_1 + \ldots + q'_{i-1} > (q_i - q'_i)$. This means that we can modify the solution found by the LP by moving $(q_i - q'_i)$ units of scheduled production from earlier batches to batch i. We still have a feasible solution since $q_1, \ldots, q_k$ are feasible. Obviously, the setup cost does not increase. Now $(q_i - q'_i)$ amount of products are produced later than before, so the holding cost must have decreased. Hence, $q'_1, \ldots, q'_k$ cannot be optimal.

Case 2: $q'_i > q_i$.

By the definition of the greedy algorithm, we produce as much as we can in period t_i without violating any of the constraints. If $q'_i > q_i$, then $q'_1, \ldots, q'_k$ cannot be feasible. ■

C.2 Genetic Algorithm

A typical genetic algorithm begins with a group of solutions to the problem, called a *population*. The particular solutions are called *individuals*. An individual usually consists of a sequence of 0's and 1's that characterizes the solution. (Not surprisingly, the 0's and 1's are called *genes*.) Our model is somewhat atypical in that individuals consist of sequences of batch due dates; in other words the i-th gene contains the due date of the i-th batch. All individuals have the same number of genes in a given run of the algorithm.

The initial population is referred to as the first *generation*. The algorithm is iterative, modifying each generation of solutions to form the next until a user-specified number of generations is reached. A new generation is formed from the previous one as follows.

First, the individuals of the old generation are evaluated using the objective function. A few of the best individuals of the old generation are added to the new generation. New individuals are then generated via a mechanism called *breeding*; the algorithm selects two parent individuals from among the best of the old generation and creates a new individual by randomly choosing gene values from each of the parents. In our heuristic, the sets of parents used to create new individuals are selected with replacement from the k best individuals in the current generation, where k is a pre-specified parameter. The i-th gene (batch due date) of a new individual is chosen by randomly selecting the value of the i-th gene from one of the two parents, where the gene value from either parent is equally likely to be chosen.

New individuals are also formed through *mutation*; the algorithm selects an individual from the old generation and randomly alters some of its genes to create a new individual. Our heuristic selects m individuals for mutation (with replacement) from the n best individuals in the current generation. (m and n are pre-specified parameters.) The heuristic considers the genes of these individuals one at a time and increases, decreases, or leaves unchanged the value of each gene according to fixed probabilities.

A final mechanism is called *immigration*; the algorithm generates new individuals from scratch in the same way that it formed the initial population. All of the individuals generated via breeding, mutation, and immigration are added to the new generation. The number of new individuals is limited so that the size of the generations remains constant.

The following is a pseudocode for the genetic algorithm:

Initialize: *old_generation*

repeat

evaluate *old_generation*

initialize *new_generation*

add some good members of *old_generation* to *new_generation*

repeat

choose two good members of *old_generation*

randomly select genes from each to form a new individual

add the new individual to *new_generation*

until a fixed number of new individuals have been bred

repeat

choose a good member of *old_generation*

randomly change some of the individual's genes

add the newly created individual to *new_generation*

until a fixed number of individuals have been mutated

repeat

create a new individual from scratch

add the new individual to *new_generation*

until a fixed number of individuals have immigrated

old_generation $\leftarrow$ *new_generation*

until a fixed number of generations have been created

C.3 Simulated Annealing

The number of batch sizes determines the dimension of x. We initialize x by randomly assigning batch due dates x_i ($x_i \in \{0, 1, \ldots, H\}$) to each entry of x with the additional constraint that identical due dates for two or more entries are not allowed.

We definie a neighborhood topology in the following way. $\tilde{x}$ is a neighbor of x if and only if $\exists!\ i_0 : |\tilde{x}_{i_0} - x_i| = 1$ and $\forall\, i \neq i_0 : \tilde{x}_i = x_i$. In other words, $\tilde{x}$ is a neighbor of x if we permute exactly one entry of x by ± 1.

The following is a pseudocode for simulated annealing:

Initialize: $x, temperature$

repeat

  repeat

    $y \leftarrow perturb(x)$

    if $cost(y) < cost(x)$ **then**

      $solution \leftarrow y$

    else

      if $exp\left(\frac{cost(x)-cost(y)}{temperature}\right) > random[0,1)$ **then**

        $solution \leftarrow y$

      end if

    end if

    $x \leftarrow solution$ (update solution)

  until equilibrium is reached

  reduce $temperature$

until temperature is sufficiently low (system is "frozen")

C.4 Tabu Search

The following is a pseudocode for Tabu Search:

```
Initialize: x, tabu
repeat
   found ← 0
   repeat
      y ← neighbor of x
      if y not in tabu and cost(y) < cost(x) then
         accept y as new solution
         found ← 1
      else
         move to the next batch
      end if
      update solution x ← y
   until found = 1 or all batches have been searched
   if found = 1 then
      update tabu
   else
      diversification
   end if
until a fixed number of iterations is executed
```

C.5 Randomized Local Search

The following is a pseudocode for Randomized Local Search:

```
repeat
    randomly generate a batch schedule x
    found ← 0
    repeat
        y ← neighbor of x
        if cost(y) < cost(x) then
            accept y as new solution
            found ← 1
        else
            move to the next batch
        end if
    until found = 1 or all batches have been searched
    if found = 1 then
        update the best solution encountered
    end if
until a fixed number of iterations is executed
```

C.6 Single Machine Heuristic

Summarizing the discussion in section 5.6, we have the following algorithm:

Step 1: Modify the network of the demand machine by eliminating nodes and arcs of type (C1) and (C2).

Step 2: Solve the modified network using network simplex.

if the network problem is infeasible **then**

  if this is the first iteration **then**

    the original problem is infeasible

  else

    no feasible solution is found

  end if

  go to End of Algorithm

end if

Step 3: Check the feasibility requirements against (C3).

if the selected batches are feasible **then**

  if this is the first iteration **then**

    an optimal solution is found

  else

    a feasible solution is found

  end if

  go to End of Algorithm

else

  eliminate arcs so that (C3) does not hold

  go to Step 2

end if

End of Algorithm

C.7 Round-Off Heuristic

The following is a pseucode for the round-off heuristic:

```
i ← 1
solve the relaxation problem
if ∃ 0,1 solution then
  stop
else
  repeat
    if batch < 0.1 · i then
      batch ← 0
    else
      if batch > 0.02 · i then
        batch ← 1
      end if
    end if
    i ← i + 1
    solve the relaxation problem with fixed variables
  until ∃ 0,1 solution
end if
```

References

J. Adams, E. Balas, and D. Zawack. 1988. The Shifting Bottleneck Procedure for Job Shop Scheduling. *Management Science*, **34**, 391–401.

C. Akkan. 1996. Overtime Scheduling: An Application in Finite Capacity Real Time Scheduling. *Journal of the Operational Research Society*, **47**, 1137–1149.

K.R. Baker. 1974, *Elements of Sequencing and Scheduling.* John Wiley & Sons, New York.

J. Blazewicz, W. Domschke, and E. Pesch. 1996. The Job Shop Scheduling Problem and New Solution Techniques. *European Journal of Operational Research*, **93**, 1–33.

J.H. Bookbinder and A.I. Noor. 1985. Setting Job-shop Due Dates with Service Level Constraints. *Journal of the Operational Research Society*, **36**, 1017–1026.

T.C.E. Cheng and M.C. Gupta. 1989. Survey of Scheduling Research Involving Due Date Determination Decisions. *European Journal of Operations Research*, **38**, 156–166.

R.W. Conway, W.L. Maxwell, and L.W. Miller. 1967, *Theory of Scheduling.* Addison Wesley, Massachussets.

S. Dauzere-Peres and J.B. Lasserre. 1997. Lot Streaming in Job-Shop Scheduling. *Operations Research*, **45**, 584–595.

I. Duenyas. 1995. Single Facility Due Date Setting with Multiple Customer Classes. *Management Science*, **41**, 608–619.

I. Duenyas and W.J. Hopp. 1995. Quoting Order Lead Times. *Management Science*, **41**, 43–57.

S. T. Enns. 1996. Finite Capacity Scheduling Systems: Performance Issues and Comparisons. *Computers & Industrial Engineering*, **30**, 727–739.

S. T. Enns. 1998. Lead Time Selection and the Behaviour of Work Flow in Job Shops. *European Journal of Operational Research*, **109**, 122–136.

L. Gelders and P.R. Kleindorfer. 1974. Co-ordinating Aggregate and Detailed Scheduling Decisions in the One Machine Job Shop: Part I Theory. *Operations Research*, **22**, 46–60.

L. Gelders and P.R. Kleindorfer. 1975. Co-ordinating Aggregate and Detailed Scheduling Decisions in the One Machine Job Shop: Part II Computation and Structure. *Operations Research*, **23**, 312–324.

M. Gen and R. Cheng. 1997, *Genetic Algorithms and Engineering Design.* John Wiley & Sons, Inc.

L.S. Gould. 1998. Introducing APS: Getting Production in Lock Step with Customer Demand. *Automotive Manufacturing & Production*, **10**, 54–58.

S. Hill. 1998. SAP's Apparel/Footwear Solution: Does It Have All the Answers? *Apparel Industry Magazine*, **59**, 62–63.

C.A. Holloway and R.T. Nelson. 1974. Job Shop Scheduling with Due Dates and Overtime Capability. *Management Science*, **21**, 68–78.

W.J. Hopp and M.L. Spearman. 1996, *Factory Physics.* Irwin, Chicago.

P. Kevin. 1996. Right From the Get-go. *Manufacturing Systems*, **14**, 34.

J. Kim and Y. Kim. 1995. Simulated Annealing and Genetic Algorithms for Scheduling Products with Multi-Level Product Structure. *Computers and Operations Research*, **23**, 857–868.

J. King. 1996. Software Delivers Customized PC Orders, Tracking. *Computerworld*, **30**, 45.

E.L. Lawler, J.K. Lenstra, A.H.G. Rinnoy Kan, and D.B. Shmoys. 1993, *Sequencing and Scheduling: Algorithms and Complexity*, volume 4. Elsevier Science Publishers.

J.E. Layden. 1996. A Rapidly Changing Landscape. *Manufacturing Systems*, **March Issue A10**.

A.G. Loerch. 1990. A New Approach to Production Planning Scheduling and Due-Date Quotation in Manufacturing Systems. Ph.D. Dissertation, Cornell University, Ithaca, New York.

A.G. Loerch and J.A. Muckstadt. 1994. An Approach to Production Planning and Scheduling in Cyclically Scheduled Manufacturing Systems. *International Journal of Production Research*, **32**, 851–871.

H. Luss and M.B. Rosenwein. 1993. A Due Date Assignment Algorithm for Multiproduct Manufacturing Facilities. *European Journal of Operational Research*, **65**, 187–198.

G.L. Nemhauser and L.A. Wolsey. 1988, *Integer and Combinatorial Optimization.* John Wiley & Sons, Inc.

S.A. Slotnick and T.E. Morton. 1996. Selecting Jobs for a Heavily Loaded Shop with Lateness Penalties. *Computers and Operations Research*, **23**, 131–140.

S.G. Taylor and S.F. Bolander. 1997. Process Flow Scheduling: Past, Present and Future. *Production and Inventory Management Journal*, **38**, 21.

P.J.M. van Laarhoven and E.H.L. Aarts. 1987, *Simulated Annealing: Theory and Applications.* D. Reidel Publishing Company.

P.J.M. van Laarhoven, E.H.L Aarts, and J.K. Lenstra. 1992. Job Shop Scheduling by Simulated Annealing. *Operations Research*, **40**, 113–125.

F.A.W. Wester, J. Wijngaard, and W.H.M. Zijm. 1992. Order Acceptance Strategies in a Production-to-order Environment with Setup Times and Due-dates. *International Journal of Production Research*, **30**, 1313–1326.