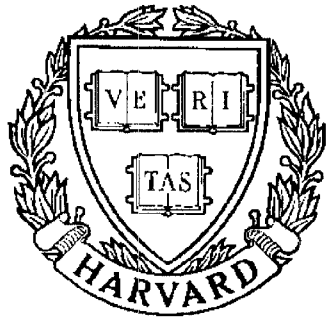
TECHNICAL RESEARCH REPORT

SYSTEMS RESEARCH CENTER



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

Optimization Based Job Shop Scheduling

by K.L. Musser, J.S. Dhingra and G.L. Blankenship

TR 91-9

Optimization Based Job Shop Scheduling*

K.L. Musser, J.S. Dhingra, G.L. Blankenship

Abstract

A generalized job shop scheduling problem is defined in detail. The proposed factory description is sufficiently realistic to model the routing and sequencing decisions made in a real manufacturing plant. An optimization problem is posed, permitting the use of very general cost functions. A variation of the method of simulated annealing is proposed as a tool for the solution of the optimization problem. A novel technique for embedding the space of feasible schedules into a permutation group is used to define a neighborhood structure for the simulated annealing process. This technique has algorithmic advantages over working directly in the space of schedules. These ideas were used to construct a scheduling software system which is in use at a Texas Instruments Custom Manufacturing Unit. We give a brief description of the software system, called ABES for Annealing Based Experiment in Scheduling, and comment on its effectiveness.

INTRODUCTION

The general job shop scheduling problem is of perennial interest because of it's direct relevance in practical manufacturing problems and because of the moderate level of progress that has been obtained toward a computationally feasible solution. A multitude of approaches to the problem have been suggested. Among these are expert system methods [2], use of neural networks methods [3], heuristics for finding good schedules in specialized problems [4, 5], and other general tools for combinatorial optimization.

Our approach falls in the last category, using a variation of one of the promising methods of general combinatorial optimization, namely simulated annealing. We have adapted it specifically to due date oriented scheduling. Although simulated annealing methods have been applied to the job shop scheduling problem [1], our approach is technically different and more general, and ours appears to be the first attempt to make use of the concept in a real manufacturing environment.

It is useful to place our work in the context of other recent modelling techniques for discrete event systems in manufacturing. The hierarchical flow control model used by Gershwin [10] assumes constant demand rates to be produced on unreliable machines. To each level in the hierarchy, events at a lower level appear deterministic and continuous. A similar manufacturing system model using constant demand rates in a multiproduct environment is analyzed in Perkins and Kumar[11]. In contrast with these our work uses

*This work was supported by NSF Engineering Research Centers Program NSFD CDR 88003012, and by a grant from Texas Instruments, Inc.

a model of discrete operations performed on various machines to meet specific customer orders.

The algebraic formalisms used to describe discrete event systems typically do not provide a natural description of the scheduling problem. Whereas the modelling techniques employed in Ramadge and Wonham [9] capture the control aspects of a scheduling decision, they do not model the timing aspects needed to evaluate a schedule. In contrast with this, the dioid algebras proposed by Cohen, et al. [8], model a discrete event system by decision free timed Petri nets. This description captures the performance aspects but does not provide a mechanism to model rerouting and sequencing decisions.

This report is organized as follows. Section I formulates a generalization of the job shop scheduling problem. This becomes the basis for our choice of algorithms in the remaining sections. Section II describes the random optimization algorithm we have used to solve the problem. We give a proof of convergence for the algorithm. Section III gives details of an implementation which has been put in place at a printed circuit board assembly factory.

I. FACTORY MODEL

We formulate the job shop scheduling problem as an optimization problem. In this section, we concentrate first on modeling the factory. Our model is a generalization of the traditional job shop [6], in which we permit multiple routes for parts and either flow or batch processing at each machine. We define the term *schedule* in this context, and state an optimization problem on the space of feasible schedules.

We model the factory as a set of physical machines $\mathcal{PM} = \{m_1, m_2, ..., m_{|\mathcal{PM}|}\}$. We define a *virtual machine* as an ordered collection of distinct physical machines, $vm_i =< m_{k_1}, m_{k_2}, ..., m_{k_{|vm_i|}} >$. We denote the set of virtual machines as $\mathcal{VM}$. A virtual machine is a group of physical machines which work as a unit, passing parts from one physical machine in the sequence to the next. Of course, it may happen that a virtual machine contains only one physical machine, in which case the physical machine does not work in concert with other physical machines. Each physical machine in $\mathcal{PM}$ may appear in many virtual machines.

We are given a set of *devices*, $\mathcal{D} = \{D_1, D_2, ...D_{|\mathcal{D}|}\}$, which can be constructed by the factory. To construct device D_i a sequence of *device operations*, $< O_k^i >_{k=1}^{N_i}$, must be performed in order. That is, operation O_k^i must begin before operation O_{k+1}^i. (The restriction to sequences of operations is for notational simplicity. There is a natural extension to general precedence relations among operations in a given device.) Let $\mathcal{O}$ denote the set of all operations, and let $\mathcal{SO}$ denote the set of finite sequences of distinct operations. We may have to schedule several orders for each device; hence, there is also a set of *batches*, $\mathcal{B} = \{b_1, b_2, ..., b_{|\mathcal{B}|}\}$. Each batch contains many instances of a single device. For each batch, b, of device i, there is a sequence of *operation instances*, denoted $< (b, O_k^i) >_{k=1}^{N_{device(b)}}$, where (b, O_k^i) represents operation O_k^i acting on all parts in batch b. We denote the set of operation instances as $\mathcal{OI}$, and the set of finite sequences of operation instances as $\mathcal{SOI}$. Further, each batch b_i has an earliest start time, a due time, and a priority chosen from a finite set $\mathcal{W}$. Below we present this formally in terms of logical relations, which are, for our purposes,

functions mapping one set into another. We define several operators to make the notation concise.

BINARY OPERATORS

$\underline{in} : \mathcal{PM} \times \mathcal{VM} \rightarrow$ { true, false }
$(m \ \underline{in} \ vm) =$ true $\Leftrightarrow vm = < k_1, k_2, ..., k_n > \bigwedge m = k_i$ for some i.

A similar definition applies for
$\underline{in} : \mathcal{O} \times \mathcal{SO} \rightarrow$ { true, false }

TERNARY OPERATORS

$(\underline{follows}, \underline{in}) : \mathcal{O} \times \mathcal{O} \times \mathcal{SO} \rightarrow$ { true, false }
$(o_i \ \underline{follows} \ o_j \ \underline{in} \ seq) =$ true $\Leftrightarrow seq = < k_1, k_2, ..., k_n > \bigwedge k_p = o_j \bigwedge k_{p+1} = o_i$ for some p.

Again, a similar definition applies for $(\underline{follows}, \underline{in}) : \mathcal{OI} \times \mathcal{OI} \times \mathcal{SOI} \rightarrow$ { true, false }
Distinction between the operators with identical names will be clear from context.

FUNDAMENTAL RELATIONS

$op\_seq : \mathcal{D} \rightarrow \mathcal{SO}$
The sequence of operations used to construct a device, subject to
$o \ \underline{in} \ op\_seq(b) \bigwedge o \ \underline{in} \ op\_seq(a) \ \Rightarrow \ a = b$

$device : \mathcal{B} \rightarrow \mathcal{D}$
The device type of a given batch.

DERIVED SET DEFINITIONS

$\mathcal{OI} \equiv \{(b, o) | b \in \mathcal{B} \bigwedge o \ \underline{in} \ op\_seq(device(b))\}$

$\mathcal{SOI} \equiv \{< k_1, k_2, ..., k_n > | k_i \neq k_j$ when $i \neq j \bigwedge k_i \in \mathcal{OI}$ for all $i, j\}$

These are *operation instances* and *sequences of operation instances*, respectively.

Associated with each operation is a set of *methods*, all of which perform the same function. Hence, the operation may be performed by following any one of the methods. For example, suppose an operation is "place surface mount ICs A-Z onto PC board," and it may be done by operator A on the Fuji machine, or by operator B on the Omni machine. The alternatives are the *available methods* for this operation. The selection of which method to use is the part of the scheduling problem referred to as *routing*. Each method has a virtual machine on which it must be performed, the *processing time* required by the method, a *principal setup time* for the virtual machine, and a so-called *method family*. If methods of the same family are performed in succession on a virtual machine, the actual time for a

setup change is a fixed fraction of the principal setup time.

There are two ways to transfer parts from one operation to the next. In one case, the entire batch is processed by the first operation before it begins at the next. In the other, parts are "pipelined" from one operation to another. In this case, the second operation begins after a short delay, called the *transfer delay.* When batch processing is used instead of "pipelining," we say the transfer delay has the value BATCH. These properties of the model are described by the additional relations:

FUNDAMENTAL BATCH and METHOD RELATIONS

$qty : \mathcal{B} \rightarrow Z^+$	$qty(b)$ is the quantity of batch b.
$earliest\_start : \mathcal{B} \rightarrow \Re$	Batch b cannot start before $earliest\_start(b)$.
$due\_time : \mathcal{B} \rightarrow \Re$	Batch b is due at $due\_time(b)$.
$priority : \mathcal{B} \rightarrow \mathcal{W}$	This is used by the cost function to be defined later.

notation:
$\mathcal{MD}$ is the set of *methods.*
$\mathcal{MF}$ is the set of *method families.*

$avail\_method : \mathcal{O} \rightarrow 2^{\mathcal{MD}}$	$avail\_method(o)$ is a nonempty subset of $\mathcal{MD}$; any of the methods can perform operation o.
$v\_mach : \mathcal{MD} \rightarrow \mathcal{VM}$	
$proc\_time : Z^+ \times \mathcal{MD} \rightarrow \Re^+$	$proc\_time(q, mthd)$ is the processing time required by method $mthd$ on a batch of quantity q.
$principal\_setup\_time : \mathcal{MD} \rightarrow \Re^+$	The time required to setup the virtual machine used by a given method.
$method\_family : \mathcal{MD} \rightarrow \mathcal{MF}$	
$transfer\_delay : \mathcal{MD} \rightarrow \{\text{BATCH}\} \bigcup \Re^+$	The transit delay from the start of an operation to the start of the next.

We are now in position to define a schedule. A *schedule* is an assignment of an allowed method to each operation instance (the routing problem), and a sequence of operation instances to each physical machine (the sequencing problem). We identify the schedule, s, with a graph, $\mathrm{G_S} = (\mathrm{N}, \mathrm{E_S})$, as follows. The vertices are all operation instances, $\mathrm{N} = \mathcal{OI}$. The set of edges, $\mathrm{E_S}$, comes from two sources. For each batch b, and each k, there is an edge in the graph from vertex $(b, O_k^{device(b)})$ to vertex $(b, O_{k+1}^{device(b)})$. For each physical machine, the sequence of operation instances assigned to it similarly defines a set of edges in the

graph. A schedule is *feasible* if the graph so generated is acyclic. The precise definition of a schedule and its graph are as follows:

DEFINITION OF SCHEDULE

A *schedule*, denoted $s = (assigned\_method, mach\_op\_seq)$, is an ordered pair of relations which satisfies the following requirements:

$assigned\_method : \mathcal{OI} \rightarrow \mathcal{MD}$
$assigned\_method((b,o)) \in avail\_method(o)$ for every $(b,o) \in \mathcal{OI}$

$mach\_op\_seq : \mathcal{PM} \rightarrow \mathcal{SOI}$
(1) for any $(b,o) \in \mathcal{OI}$, there exists $m \in \mathcal{PM}$ with (b,o) *in* $mach\_op\_seq(m)$
(2) m *in* $v\_mach(assigned\_method((b,o))) \Leftrightarrow (b,o)$ *in* $mach\_op\_seq(m)$

Notation:
$\mathcal{FMA}$ = the set of feasible method assignments.
$\mathcal{FMA} \equiv$ {all maps $\mu : \mathcal{OI} \rightarrow \mathcal{MD}$ such that $\mu((b,o)) \in avail\_methods(o)$ for any $(b,o) \in \mathcal{OI}$}
$\mathcal{S}$ = the set of feasible schedules.
$\mathbf{N} = \mathcal{OI}$ = set of vertices
$\mathbf{E_s} \subseteq \{(u,v)$, where $u, v \in \mathcal{O}\}$ = directed edges for schedule s.

IDENTIFICATION OF A SCHEDULE WITH A DIRECTED GRAPH

The graph $\mathbf{G_s} = (\mathbf{N}, \mathbf{E_s})$ associated with schedule s satisfies:

(1) for $(b,o),(b,p) \in \mathcal{OI}, p$ *follows* o *in* $op\_seq(device(b)) \Longrightarrow ((b,o),(b,p)) \in \mathbf{E_s}$
(2) for $m \in \mathcal{PM}, (b^1,o^1),(b^2,o^2) \in \mathcal{OI}$,
(b^2,o^2) *follows* (b^1,o^1) *in* $mach\_op\_seq(m) \Rightarrow ((b^1,o^1),(b^2,o^2)) \in \mathbf{E_s}$
(3) Every edge in $\mathbf{E_s}$ is obtained from either (1) or (2)

The times when operation instances begin and end can be computed from the schedule. Each operation is assumed to begin as soon as all its preceding operations have taken place. The algorithm for computing these times is presented in the appendix. A performance measure for a schedule can then be computed from the start and finish times. Any performance measure which is computable from the start and finish times of all operations is permitted. Our optimization problem is then to maximize the performance measure over the set of feasible schedules, or, equivalently, to minimize the associated cost.

$\mathcal{TV}$ = { all maps $\mathcal{OI} \to \Re$} NOTE: $\mathcal{TV}$ stands for Time Vector.

$start\_times : \mathcal{S} \to \mathcal{TV}$

$finish\_times : \mathcal{S} \to \mathcal{TV}$

$c : \mathcal{TV} \times \mathcal{TV} \to \Re$ This is the cost function.

For notational simplicity, we write $c(s)$ for $c(start\_times(s), finish\_times(s))$. We then have the following optimization problem.

$$x^\star = \arg\min_{s \in \mathcal{S}} c(s)$$

II. OPTIMIZATION ALGORITHM

In operations scheduling, performance measures of interest include due date performance, work in process inventories, and machine utilization. We permit a cost function including each of these elements, and we use a variation of "simulated annealing" to optimize it. The scheduling problem we consider has two related but distinguishable components, namely routing and sequencing. For this reason a two level stochastic descent method is natural. In the following paragraphs we describe the two level algorithm, and we justify our claim of convergence to a global optimum.

In combinatorial optimization, one wishes to find one of a finite number of items which has lowest cost. There may be more than one item with this cost, and in that case any of them will do. In our application, the items considered are the possible schedules, of which there is a finite, but large, number. Simulated annealing treats each schedule as a state in a discrete time finite state Markov chain. The transition probabilities of the Markov chain are selected so that the state drifts toward the lower cost schedules.

The simulated annealing algorithm uses a Markov process originally described by Metropolis[14]. An initial state (schedule), X_0, is chosen at random. Next, a nearby "trial" state, X_1^{trial}, is chosen by some probabilistic rule, and the two are compared. If the "trial" state has lower cost, it is accepted, with probability 1, as the new state of the Markov chain. Otherwise it is accepted with a lower probability determined by the relative costs. More precisely,

$$\begin{aligned} \Pr\{X_{k+1} = X_{k+1}^{trial} | X_{k+1}^{trial}, X_k\} &= \begin{cases} 1 & \text{if } c(X_{k+1}^{trial}) < c(X_k) \\ e^{-[c(X_{k+1}^{trial}) - c(X_k)]/T_k} & \text{otherwise} \end{cases} \\ \Pr\{X_{k+1} = X_k | X_{k+1}^{trial}, X_k\} &= 1 - \Pr\{X_{k+1} = X_{k+1}^{trial} | X_{k+1}^{trial}, X_k\} \end{aligned}$$

Here the value, T_k is called the "temperature," and the sequence $\{T_k\}_{k=0}^{\infty}$ is called the temperature schedule. If the trial state is only accepted when it has lower cost, the algorithm will remain in any local minimum it may find. This is the case when the temperature is zero. A nonzero temperature, however, gives the Markov chain the opportunity to escape from local minima. We will refer to a Markov process evolving in this manner as a Metropolis

process. Simulated annealing is a special case in which $T_k \rightarrow 0$ slowly, so that the state of the Markov chain converges in probability to the collection of states with globally minimum cost[13]. In practice, since we can keep track of the best state visited so far, we are more interested in having ever reached a minimum than reaching it and staying there.

Application of simulated annealing is the art of creating the random variables, X_k^{trial} and X_0, and selecting the temperature schedule so that convergence to a global minimum is as rapid as possible. We concentrate our discussion on selecting X_k^{trial} and the temperature schedule. For a discussion of the initial schedule, X_0, see Section III on implementation.

As described earlier, a schedule consists of a method assignment and a sequence of operation instances for each physical machine. It is algorithmically and conceptually simpler to consider an alternative representation for the second part. Instead of using a separate sequence of operation instances for each machine, we use a single sequence containing all the operation instances. The space, $\mathcal{S}$, of feasible schedules is, then, replaced by $\mathcal{S}_{new} \equiv \mathcal{FMA} \times S_n$, where S_n is the group of permutations of $n = |\mathcal{OI}|$ operation instances, and $\mathcal{FMA}$ is the feasible method assignments. Our new representation for a schedule $s = (assigned\_method, mach\_op\_seq)$ is $s_{new} = (assigned\_method, \sigma)$, where $\sigma \in S_n$ is a sequence of the operation instances. (The sequences of n distinct operation instances are used interchangeably with the permutations, where an arbitrary sequence is equated to the identity permutation.)

We can determine σ from s, albeit nonuniquely, as follows. Form the graph $\mathbf{G_s}$ for s. Let α be an empty string of vertices of $\mathbf{G_s}$. Since s is feasible, $\mathbf{G_s}$ is acyclic, and there is a vertex which has no incoming arcs. Delete this vertex from the graph, and append it to the end of α. Also delete all of its outgoing arcs. The graph remaining is still acyclic. This process may be repeated until all vertices have been transferred to α. Now the sequence in α is a permutation of the elements of $\mathcal{OI}$, with the property that if there was a path from vertex A to vertex B in $\mathbf{G_s}$, then A appears earlier in α than does B.

The sequence of operation instances for a given batch, b, may be found from such a permutation by deleting from the list those operation instances belonging to a different batch. We call the sequence remaining the *filter* of α onto batch b, and denote this $filter_b(\alpha)$. Note that we must have:

$$filter_b(\alpha) = < (b, O_k^{device(b)}) >_{k=1}^{N_{device(b)}}$$

That is, the order of the operation instances must match that specified by $op\_seq(device(b))$. Similarly, the sequence of operation instances for physical machine m may be found by deleting any operation instance (b, o) for which

$$\neg\ m\ \underline{in}\ v\_mach(assigned\_method((b, o)))$$

Hence, a well-formed element of $\mathcal{S}_{new}$ identifies a unique schedule. To permit the use of all permutations, not just those which are well-formed, we define equivalence classes as follows.

Definition: Let $\sigma_1 = < (b_1^1, o_1^1), (b_2^1, o_2^1), ..., (b_n^1, o_n^1) >$ and $\sigma_2 = < (b_1^2, o_1^2), (b_2^2, o_2^2), ..., (b_n^2, o_n^2) >$ be permutations of $\mathcal{OI}$. We say σ_1 and σ_2 are *H-equivalent* if $b_i^1 = b_i^2$ for all i.

Proposition 1 *Let e_{S_n} be the identity permutation. Then the set $H \equiv \{\sigma \in S_n | \sigma$ is* H-equivalent to $e_{S_n}\}$ *is a subgroup of S_n. Furthermore, for any $\pi \in S_n$, $\{\sigma \in S_n | \sigma$ is* H-equivalent to $\pi\} = H\pi$, *where $H\pi$ is a right coset of H in S_n.*

Proof:

The group, S_n, of permutations of $\mathcal{OI}$ is the set of invertible maps, $\sigma : \mathcal{OI} \rightarrow \mathcal{OI}$, where the group operation is composition of functions. The identity element is the identity map, $e_{S_n}((b,o)) = (b,o)$. Then σ_1 and σ_2 are H-equivalent iff $\forall (b,o) \in \mathcal{OI}$, $\sigma_1((b,o)) = (a_1,o_1) \bigwedge \sigma_2((b,o)) = (a_2,o_2) \Rightarrow a_1 = a_2$. Hence σ is H-equivalent to e_{S_n} (i.e. $\sigma \in$H) iff $\forall (b,o) \in \mathcal{OI}, \sigma((b,o)) = (b,o^2)$ for some o^2. We must show that H is closed under composition and under inverses. Let $\sigma_1, \sigma_2 \in$H. Then $\forall (b,o) \in \mathcal{OI}, (\sigma_2 \circ \sigma_1)((b,o)) = \sigma_2(\sigma_1((b,o))) = \sigma_2((b,o^2)) = (b,o^3)$ for some o^3. Therefore, $(\sigma_2 \circ \sigma_1) \in$H. H is also clearly closed under inverses since $\sigma((b,o^1)) = (b,o^2) \Leftrightarrow (b,o^1) = \sigma^{-1}((b,o^2))$. Therefore, H is a subgroup of S_n.

For the second part, we need only to show that two maps, π_1 and π_2 are H-equivalent iff $\pi_1 \circ \pi_2^{-1} \in$H. Now π_1 and π_2 are H-equivalent $\Longleftrightarrow \forall (b,o) \in \mathcal{OI}$, $\exists b^1, o^1, o^2$ such that $\pi_1((b,o)) = (b^1,o^1)$ and $\pi_2((b,o)) = (b^1,o^2)$.$\Longleftrightarrow \forall (b^1,o^2) \in \mathcal{OI}$, $(\pi_1 \circ \pi_2^{-1})((b^1,o^2)) = \pi_1((b,o)) = (b^1,o^1)$. $\Longleftrightarrow (\pi_1 \circ \pi_2^{-1}) \in$H. □

There is exactly one permutation, σ, in each coset which satisfies

$$filter_b(\sigma) = < (b, O_k^{device(b)}) >_{k=1}^{N_{device(b)}}$$

for all $b \in \mathcal{B}$. We call this permutation the *canonical representative* of the coset. Let μ be any valid method assignment (i.e. satisfying $\mu((b,o)) \in avail\_method(o)$ for all $(b,o) \in \mathcal{OI}$), and π any permutation in S_n. Then we can associate with $(\mu,\pi) \in \mathcal{S}_{new}$ the schedule determined by (μ,σ), where σ is the canonical representative of $H\pi$.

The set of valid method assignments, $\mathcal{FMA}$, is itself a group under a properly defined operation. For any $o \in \mathcal{O}$, the set $avail\_methods(o)$ is a finite set. If its elements are given an order, then it is a group under addition modulo $|avail\_methods(o)|$. Similarly, $\mathcal{FMA}$ is a group under the operation given by

$$(\mu_1 \oplus \mu_2)((b,o)) = \mu_1((b,o)) + \mu_2((b,o)) \pmod{|avail\_methods(o)|}.$$

We have thus shown the following fact.

Proposition 2 *The set $\mathcal{S}_{new}$ is a group under the operation $(\mu_1,\sigma_1) \otimes (\mu_2,\sigma_2) = (\mu_1 \oplus \mu_2, \sigma_1 \circ \sigma_2)$.*

The Markov chain $< X_k >_{k=0}^{\infty}$ has $\mathcal{S}_{new}$ as its state space. We call $< X_k >_{k=0}^{\infty}$ the *high level* Metropolis process, and we define a so-called *low level* process to generate the random variable X_{k+1}^{trial}. Let $\{g_i, i = 1, ..., N_g\}$ be a subset of $\mathcal{FMA}$ which generates the group with $g_i = e_{\mathcal{FMA}}$ for some i. Also, for any i, there is a k such that $g_i^{-1} = g_k$. We define the random variables V_{high}^k and $Y_0^k = X_k \otimes V_{high}^k$, where $V_{high}^k \in \{(g_i, e_{S_n}), i = 1, ..., N_g\}$ with probability 1, $\Pr\{V_{high}^k = (g_i, e_{S_n})\} > 0$ for each i, and the random variables V_{high}^k are i.i.d.

Thus Y_0^k has a different method assignment (route), but it has the same permutation of $\mathcal{OI}$ as X_k.

Suppose $Y_0^k = (\mu_k, \sigma_k)$. We let the "low level" Metropolis process, $< Y_i^k >_{i=0}^N$, evolve on the set $\{(\mu_k, \pi) | \pi \in S_n\}$. That is, for fixed k the method assignment is the same for all Y_i^k, but the permutation of $\mathcal{OI}$ changes. Let $\{\pi_j, j = 1, ..., N_\pi\}$ be a generating set for S_n, with $\pi_j = e_{S_n}$ for some j, requiring as above that for any j, there is a k with $\pi_j^{-1} = \pi_k$. Then we have $Y_{i+1}^{k,trial} = Y_i^k \otimes V_{low}^{k,i}$, where $V_{low}^{k,i} \in \{(e_{\mathcal{FMA}}, \pi_j), j = 1, ..., N_\pi\}$ with probability 1, $\Pr\{V_{low}^{k,i} = (e_{\mathcal{FMA}}, \pi_j)\} > 0$, for all j, and the random variables $V_{low}^{k,i}$ are i.i.d. We write $< T_i^k >_{i=0}^N$ for the temperature schedule associated with the low level process $< Y_i^k >_{i=0}^N$. Finally, we set $X_{k+1}^{trial} = Y_N^k$.

Proposition 3 *If $T_k = T_{high} > 0$, for all k, and $T_i^k = T_{low} > 0$, for all k, i, then for any $s_{new} \in \mathcal{S}_{new}$, $\Pr\{X_k \neq s_{new}$ for all $k \leq n\} \to 0$ exponentially fast as $n \to \infty$. If, instead, $T_k = \frac{A}{\log(k+k_0)}$, for some $k_0 \geq 1$ and all k, then for A sufficiently large, $\Pr\{c(X_n) = c(x^*)\} \to 1$ as $n \to \infty$.*

Proof:

For the first part observe that with constant temperature schedules, the Markov chain is time homogeneous. Hence it suffices to show that there is a path with positive probability from any state to any other state. Now $\Pr\{X_{k+1} = X_k \otimes v\} > 0$ for any $v = (g_i, \pi_j)$. The conclusion holds since $\{(g_i, \pi_j); i = 1, ..., N_g; j = 1, ..., N_\pi\}$ generates $\mathcal{S}_{new}$.

The second part follows directly from theorem 1 of [12], where irreducibility holds because $\{(g_i, \pi_j); i = 1, ..., N_g; j = 1, ..., N_\pi\}$ generates the set $\mathcal{S}_{new}$, and weak reversibility holds because if $\Pr\{X_{k+1} = X_k \otimes (g_i, \pi_j)\} > 0$, then $\Pr\{X_{k+1} = X_k \otimes (g_i^{-1}, \pi_j^{-1})\} > 0$. That is, every path is reversible. □

A systematic procedure for selection of the generating sets $\{g_i\}$ and $\{\pi_j\}$, and the assignment of probabilities for V_{high}^k and $V_{low}^{k,i}$ will be considered in future work. For now, we permit each g_i to differ from the identity on $\mathcal{FMA}$ in no more than one operation instance. That is, we change the routing for no more than one operation instance at a time. As for the permutation generators, $\{\pi_j\}$, we include any that can be formed as follows. Select distinct integers, i and j, (with $i < j$) between 1 and $n = |\mathcal{OI}|$, inclusive. Then π can be the permutation obtained by reversing the subsequence of elements from the i^{th} to the j^{th} position. Since all adjacent transpositions are included, the set generates S_n. We also include those permutations that can be formed by translating such a subsequence from one position in the whole sequence to another.

III. IMPLEMENTATION

The descriptive language and factory model developed in the previous sections were implemented in a software package named $\mathcal{ABES}$ (*Annealing Based Experiment in Scheduling*). This software has been installed at a Texas Instruments PC board manufacturing

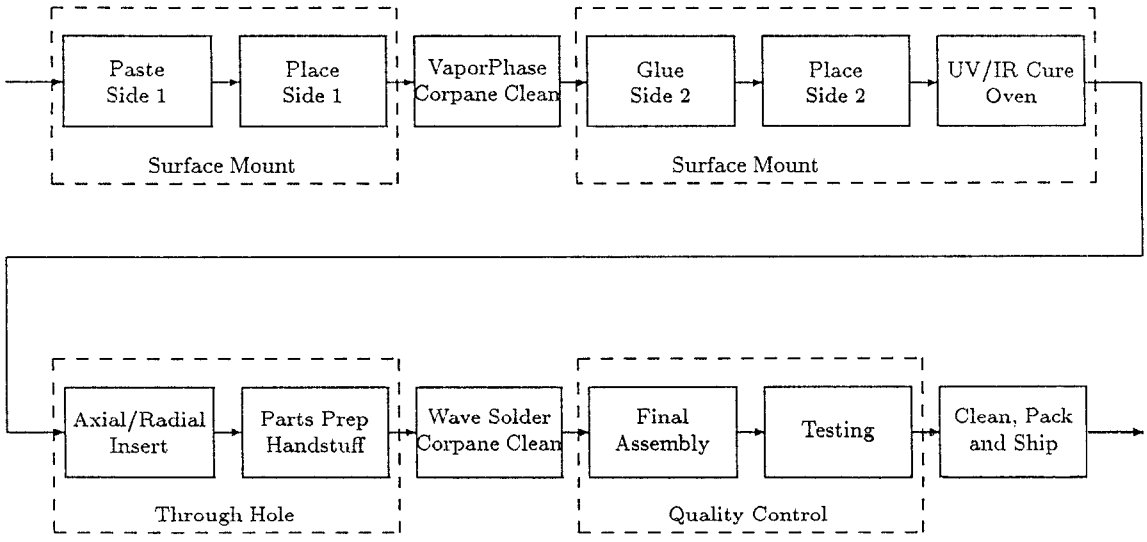


Figure 1: A typical process flow

facility in Johnson City, TN and is currently being used to schedule the surface mount machines of the Custom Manufacturing Unit of that facility.

The Custom Manufacturing Unit consists of three major sub-units, namely surface mount, through-hole and quality control. A typical process flow for a device is shown in Figure 1. The surface mount sub-unit is generally the bottle neck region in the device process flow, so the main effort was directed towards optimal scheduling of the surface mount machines.

The Custom Manufacturing Unit produces roughly 50 different devices (or boards). Depending on the demand and parts availability it handles 30 different orders or batches per week on average. The number of operations to be scheduled for each batch varies from 3 to 6 depending on whether the particular device is a single sided or double sided PC board. Therefore in principle we have an average of 100 operations to be scheduled in a week in the whole Custom Manufacturing Unit. Since the surface mount is the bottleneck region, the scheduling software was implemented to schedule the surface mount unit only. Depending on the number of sides to be processed for a particular device, only 1 or 2 operations have to be scheduled for a particular batch. Hence on an average 40-50 operations have to scheduled in the surface mount unit.

The surface mount area is divided into six machine cells called SMT lines. Each line consists of a screenprinter and component placement machines. Using the processing and

setup times of the bottleneck physical machine as the processing parameters for the cell, each cell can be modelled as a virtual machine. This allows for a continuous process flow representation (pipelining) while maintaining a batch flow among the sub-units. To allow for a similar mix of continuous and batch process flow in the surface mount sub-unit, concurrent operation scheduling with a fixed transit delay, is allowed for adjacent operations of the same batch.

The software package $\mathcal{ABES}$ is written in C and runs on a desktop computer. A menu driven user interface leads the user through a sequence of instructions for optimal schedule generation. Before a schedule is generated the principal process flow (*device operations* $< O_k >_{k=1}^{N}$) for each device has to be defined in the system. The user can also define alternate process flows, which are combinations of the *available methods* for different operations in the principal process flow. A graphical interface allows the user to add devices, remove devices or to change device processing parameters. The concept of *method families* has also been implemented in this package. The user can define the family structure along with device parameters in the initial setup. If the two successive operations scheduled on a particular machine belong to the same *method family*, the setup time on the succeeding operation is reduced to a fraction of the original setup time. This is a common feature in manufacturing systems; it occurs when the two devices have similar sets of parts to be used on a particular machine and hence only a partial setup is required. This feature permits lower downtime of the machines and leads to a lower cost schedule. During optimization the swapping of operations does lead to grouping of like devices on the same machine.

Once all devices have been defined in the system the user can generate a schedule by adding batches (lots) for particular devices. The program generates an initial schedule, X_0, on the basis of the addition of new batches to the system. For each new batch the user is prompted for the following attributes, namely, earliest start time, due time and a priority level. The *operation instances* for the new batch are inserted in the machine-idle slots in the schedule for an earliest possible finish time. This along with the delayed concurrency leads to an efficient utilization of the machines.

The trial schedules $\{X_k^{trial}\}_{k=1}^{\infty}$ are generated either by swapping adjacent *operation instances* on a single machine or by selecting a particular batch at random and interchanging each *operation instance* of the batch with the succeeding *operation instance* on the same machine. At each iteration, the choice of method for trial schedule generation is random. A trial schedule is accepted or rejected with certain probability (as defined in Section II). The acceptance probability is a function of the "annealing temperature". We use an adaptive temperature scheduling scheme in the implementation. The new temperature at each step is calculated as a function of the differential cost average over the previous iterations. This sequence of schedules constitute the so called "low level" Metropolis process. For the "high level" Metropolis process, after every 100 iterations of the low level process, a new schedule is generated by choosing a batch at random and changing its process flow (replace the principal process flow by a randomly chosen process flow from the set of alternate flows for that particular device), subject to certain constraints. The new schedule is accepted or rejected with a certain probability defined in terms of the cost differential from the previous "high level" Metropolis iteration. Here also we use adaptive temperature scheduling, although with a different temperature variable. Thus effectively we have two temperature variables

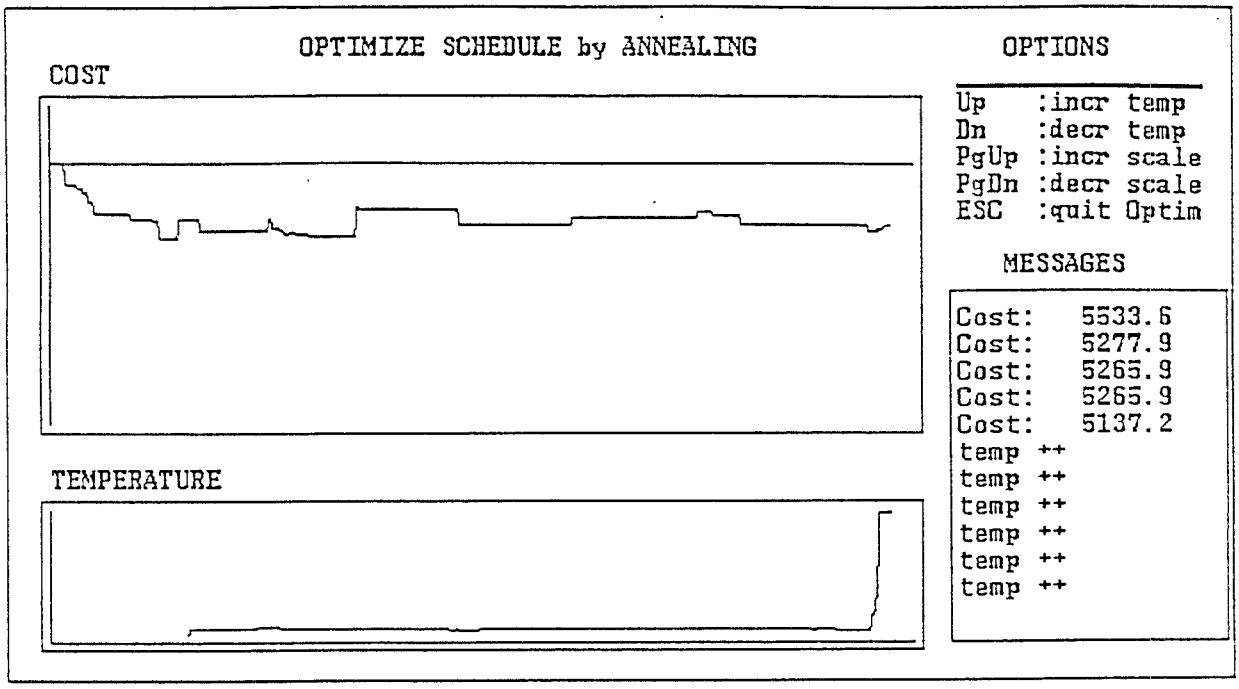


Figure 2: Optimization process

for the two level Metropolis process. Figure 2 shows the optimization process; the "high level" Metropolis iterations are shown by arrows.

Each batch in the schedule has a cost associated to it. This batch cost is calculated by adding up the three cost components, namely Work in process (WIP), Tardiness (TARD) and the Inventory (INV) cost and are calculated using the following formulas:

(WIP) (FinishTime - StartTime)*(a + b*BatchQty)

(TARD) $(a + b*T+c*T^2) * (d + e*\text{BatchQty})$
Here T = max(0, FinishTime - DueTime)

(INV) $(a + b*E+c*E^2) * (d + e*\text{BatchQty})$
Here E = max(0, DueTime - FinishTime)

The user has five different priority levels he can assign to the new batches being added to the schedule. Each priority level is defined by the user by assigning different set of values to the parameters (a, b, c, d and e) for each of the cost components. The total schedule cost, defined as the sum of the costs associated to each batch in the schedule, is used as the optimization criterion. The user has the option to enable or disable the three components globally for the purpose of evaluation of the total cost.

Since this is not an online real-time scheduler, each generated schedule has an active time associated to it. This active time represents the time origin for operation scheduling. The user has the option of specifying the active time when generating a new schedule. Once an optimal schedule has been generated, the user can update the factory status by changing this active time and also update the operation status for the different machines. Also new batches can be added to the schedule as the active time changes. The optimization process is carried out after each such update. If updates are made frequently enough this will lead to a reactive real-time scheduler.

Once an optimal schedule has been generated the user can view or print the schedule. Fig 3 shows a sample device schedule generated using $\mathcal{ABES}$. It displays the schedule for the various batches of the same device. Fig 4 displays the various batches scheduled on a particular machine. The user can view the machine utilization on a GANTT chart. Another graphical display depicts the Work-in-Process(WIP). A detailed description of $\mathcal{ABES}$ is available in [16]. In terms of the time involved for scheduling, it takes a trained user about 10 minutes to generate the initial schedule. For a typical work load of about 30 batches, the optimization procedure takes about 25 minutes for 5000 iterations on a 386SX 16MHz PC supported with a 80387 math co-processor. Therefore it takes about 35 minutes to generate a near optimal schedule. If the optimization procedure is stopped midway, the best schedule obtained till that point is displayed. Thus a sub-optimal schedule is available at each iteration and the user can stop the optimization at any point to get the best schedule found.

The present system was installed in the Texas Instruments plant in Johnson City and is currently under evaluation. The initial response to the system has been quite favorable. Mr. Larry Ferguson who has been using $\mathcal{ABES}$ for the past few months had this to say:

> Our prior method of scheduling our surface mount lines required gathering the current production status, future requirements from several scheduling groups, and any new additions or changes to process flow for each device. The new scheduling system has turned a complex, manual task into a more organized automated approach. By loading the attributes, by device, we have eliminated many of the scheduling errors associated with timing and flow. The new system does an excellent job of searching out the optimum schedule to help us increase our throughput and minimize any delinquencies.

Conclusions

A real manufacturing environment involves intricacies which are sometimes not reflected in the mathematical programming abstractions made from them. We have posed a scheduling problem which embodies product families, batch and/or flow processing, and the possibility of routing to any of several machines which are not identical. By allowing the use of a general cost function we can reflect the conflicting goals of due date performance and low work in process inventories. The simulated annealing algorithm gives a progressively improving solution, which converges to an optimal schedule. This permits an implementation

```
**************************************************************************
************************** DEVICE 1 ******************************
**************************************************************************
Operation  Machine  Setup  Batch  Start                Finish
   ID        ID     Time    Qty   Time                 Time
--------------------------------------------------------------------------
--------------------------------------------------------------------------
Batch ID: 001
 0380   SMT 4        24    800    18:00:00 (08/24)    20:40:00 (08/27)
                                          DUE:       23:00:00 (08/24)
Batch ID: 007
 0256      (parts)    0   1500                        18:00:00 (08/24)
 0257   SMT 1       150   1500    21:44:43 (08/31)    09:27:34 (09/04)
 0262   SMT 4        24   1500    22:00:00 (09/02)    00:00:00 (09/05)
                                          DUE:       23:00:00 (08/25)
Batch ID: 006
 0251      (parts)    0   1000                        10:00:00 (08/25)
 0252   SMT 1       150   1000    08:27:43 (08/28)    08:16:17 (08/29)
 0253   SMT 5        24   1000    04:11:28 (08/29)    13:31:28 (08/30)
                                          DUE:       23:00:00 (08/29)
```

Figure 3: Schedule by Device

```
*************************************************************************
*************************** MACH: SMT 2     ***************************
*************************************************************************
Operation Job  Setup  Batch  Batch    Start              Finish
   ID          Time    ID     Qty     Time               Time
*************************************************************************
  0314  Dev 6  180     017     422    18:00:00 (08/24)   00:36:53 (08/27)
  0377  Dev 2  120     029     254    02:36:53 (08/27)   06:17:37 (08/27)
  0392  Dev 5  360     002      63    08:05:37 (08/27)   17:05:37 (08/27) L
        DUE:23:00:00 (08/25)
  0337  Dev 4  240     022     200    21:05:37 (08/27)   02:00:51 (08/28)
  0374  Dev 7  210     010    1000    05:30:51 (08/28)   14:50:51 (08/29) L
        DUE:23:00:00 (08/27)
  0341  Dev 4  240     023      84    18:50:51 (08/29)   20:54:51 (08/29)
  0290  Dev 6  180     012     800    23:54:51 (08/29)   08:17:43 (08/31)
  0345  Dev 7  240     024     119    12:17:43 (08/31)   15:13:23 (08/31)
  0272  Dev 3  210     009     800    08:49:03 (09/03)   11:29:03 (09/04) L
        DUE:23:00:00 (08/31)
```

Figure 4: Machine Schedule

to be flexible by interactively using all the computer time available, but not more. Because it is based on "nearby" schedules, in the sense of neighborhoods, it can nicely handle the frequent modifications and schedule deviations which occur in an actual manufacturing environment. The experience of the Johnson City Texas Instruments factory suggests the feasibility of the method as a full scale scheduling system.

Research in several related fields may be useful in improving a system level approach to the scheduling problem. In particular, application of relational database technology will help coordinate the usually large volume of data associated with monitoring the status of a factory. Such coordination is essential if closed-loop reactive scheduling is to become a reality. Also, parallel and distributed algorithms for optimization will help expedite solving this computationally intensive problem. Finally, a cooperative application of expert systems and optimization based scheduling may prove to be the best compromise, using the stronger points of both approaches.

Acknowledgement: We gratefully acknowledge the assistance and information provided by Texas Instruments, Inc. Special thanks to Gene Lamm, Larry Ferguson, Andy Lobsenz, Barbara Papas, Richard Herrod, and Margaret Berbari.

Appendix

The following pseudo-code is the algorithm which, given a feasible schedule s, computes start and finish times for all operation instances. We explicitly model virtual machine setup times. Additionally, the processing intervals for successive operations may overlap, as when parts can be transferred between machines in quantities smaller than the batch quantity. This is modeled by the function $transfer\_delay$, whose value is either BATCH or a time interval.

First, we define two intermediary functions called $transfer\_time$ and $setup\_time$. These are derived from the relations presented in the text.

$transfer\_time : \mathcal{OI} \rightarrow \Re^+$
algorithm $tranfer\_time((b, o))$
    if $transfer\_delay(assigned\_method((b, o))$ =BATCH
        return $proc\_time(qty(b), assigned\_method((b, o)))$
    else
        return $transfer\_delay(assigned\_method((b, o)))$
    endif
end_algorithm

$setup\_time : \mathcal{OI} \times \mathcal{OI} \rightarrow \Re^+$
algorithm $setup\_time((b_1, o_1), (b_2, o_2))$
    $mthd_i = assigned\_method((b_i, o_i)$, for $i = 1, 2$.
    if $v\_mach(mthd_1) \bigcap v\_mach(mthd_2) = \emptyset$
        return 0

elseif $mthd_1 = mthd_2$
return 0
elseif $method\_family(mthd_1) = method\_family(mthd_2)$
return β $principal\_setup\_time(mthd_2)$ /* for some β, $0 < \beta < 1$ */
else
return $principal\_setup\_time(mthd_2)$
endif
end_algorithm

$start\_times : \mathcal{S} \rightarrow \mathcal{TV}$
$finish\_times : \mathcal{S} \rightarrow \mathcal{TV}$

These are computed recursively as follows:

Let $\mathbf{E_s}$ be the set of edges of the graph associated with schedule s.

if $\{(b_2, o_2) \in \mathcal{OI} | ((b_2, o_2), (b, o)) \in \mathbf{E_s}\} = \emptyset$
$(start\_times(s))((b, o)) = earliest\_start((b, o))$
else
$(start\_times(s))((b, o)) =$

$$\max_{\{(b_2,o_2)|((b_2,o_2),(b,o))\in \mathbf{E_s}\}} \begin{cases} (start\_times(s))((b_2, o_2)) + transfer\_time((b_2, o_2)) & \text{if } b_2 = b \\ (finish\_times(s))((b_2, o_2)) + setup\_time((b_2, o_2), (b, o)) & \text{otherwise} \end{cases}$$

$(finish\_times(s))((b, o)) =$
$\max\{(start\_times(s))((b, o)) + proc\_time(qty(b), assigned\_method((b, o))),$
$\max_{\{(b,o_2)\in\mathcal{OI} | o \ \underline{follows}\ o_2 \ \underline{in}\ op\_seq(device(b))\}}(finish\_times(s))((b, o_2))\}$

References

[1] P.J.M. van Laarhoven, E.H.L. Aarts, and J.K. Lenstra, "Job Shop Scheduling by Simulated Annealing," Philips Research Laboratories, Eindhoven, preprint, 1988.

[2] P.S. Ow, and S.F.Smith, "Viewing Scheduling as an Opportunistic Problem Solving Process," in R.G.Jeroslow(ed.) *Annals of Operations Research: Approaches to Intelligent Decision Support.* Baltzer Scientific Publishing Co., 1987.

[3] W. Jeffrey and R. Rosner, "Optimization Algorithms — Simulated Annealing and Neural Network Processing," *Astrophysical Journal, Part 1*, Vol. 310, 1 Nov 1986.

[4] M. Salimian, *A New Algorithm for the Three Machine Flow Shop Problem*, PhD Dissertation, *The University of Oklahoma*, 1988.

[5] M.J. Maddox, *Scheduling a Stochastic Job Shop to Minimize Tardiness Objectives*, PhD Dissertation, *The University of Michigan*, 1988.

[6] K.R. Baker, *Introduction to Sequencing and Scheduling*, J.Wiley & Sons, New York, 1974.

[7] Dempster M.A.H., J.K. Lenstra, and A.H.G. Rinnoy Kan, eds., *Deterministic and Stochastic Scheduling; Proceedings of the NATO Advanced Study and Research Institute on Theoretical Approaches to Scheduling Problems*, D. Reidel Publishing Co., 1982.

[8] G. Cohen, P. Moller, J.P. Quadrat, and M. Viot, "Algebraic Tools for the Performance Evaluation of Discrete Event Systems," *Proceedings of the IEEE*, pp. 39-58, Jan. 1989.

[9] P.J.G. Ramadge, and W.M. Wonham, "The Control of Discrete Event Systems," *Proceedings of the IEEE*, pp. 81-98, Jan. 1989.

[10] Gershwin, S.B., "Hierarchical Flow Control: A Framework for Scheduling and Planning Discrete Events in Manufacturing Systems," *Proceedings of the IEEE*, pp. 195-209, Jan. 1989.

[11] J. Perkins, and P.R. Kumar, "Stable, distributed, real-time scheduling of flexible manufacturing/assembly/disassembly systems," *Proc. 27th IEEE Conf. on Decision and Control*, Austin, TX, Dec. 1988.

[12] B. Hajek, "A Tutorial Survey of Theory and Applications of Simulated Annealing," 24^{th} *IEEE Conference on Decision and Control*, Fort Lauderdale, FL, Dec. 1985.

[13] D. Mitra, F. Romeo, and A. Sangiovanni-Vincentelli, "Convergence and Finite-Time Behavior of Simulated Annealing," 24^{th} *IEEE Conference on Decision and Control*, Fort Lauderdale, FL, Dec. 1985.

[14] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, "Equations of State Calculations by Fast Computing Machines," *J. Chem. Phys.*, 21, pp. 1087-1091, 1953.

[15] J. Blazewicz, "Selected Topics in Scheduling Theory," *Annals of Discrete Mathematics*, 31, pp. 1-60, 1987.

[16] J.S. Dhingra, Userś Guide to ABES, 1990, unpublished.