

Tradeoffs During Scheduling Of Cyclic Production Systems

Jeffrey W. Herrmann

*Department of Mechanical Engineering and Institute for Systems Research,
University of Maryland, College Park, MD 20742, USA*

Fabrice Chauvet

INRIA-Lorraine, 4 rue Marconi, 57070 Metz, France

Jean-Marie Proth

*INRIA-Lorraine, 4 rue Marconi, 57070 Metz, France,
and Institute for Systems Research, University of Maryland, College Park, MD 20742, USA*

Abstract

A cyclic production system manufactures a set of products at a constant frequency. Each resource repeatedly performs a certain set of tasks, and the system produces a certain set of units each cycle. The production schedule defines the timing of the tasks within each cycle, which determines system performance on measures such as throughput and work-in-process inventory. This paper describes the tradeoffs that occur when scheduling such systems. This paper presents algorithms for constructing feasible cyclic schedules that maximize throughput, minimize WIP, or balance the two objectives. In addition, the paper describes a problem space search that explores the tradeoffs between throughput and inventory. An experimental study shows that the scheduling algorithms generate high throughput, moderate WIP solutions. The tradeoff exploration search provides a series of solutions that allow a decision-maker to tradeoff the WIP and throughput objectives.

Please send correspondence to
Jeffrey W. Herrmann
Department of Mechanical Engineering
University of Maryland
College Park, MD 20742
301.405.5433 TEL
301.314.9477 FAX
jwh2@eng.umd.edu

1. Introduction

A cyclic production system manufactures a set of products at a constant frequency. That is, each resource repeatedly performs a certain set of tasks, and the system produces a certain set of units each cycle.

This paper addresses the following type of cyclic production system. The set of tasks does not change, the tasks are non-preemptive, and the task processing times are deterministic. Each task requires exactly one resource, and there are sufficient buffers to hold parts that are not being processed. All resources are perfectly reliable, and each can process at most one task at a time. The processing requirements of different products impose task precedence constraints, and the corresponding sequences of resources may be different for each product. (That is, the system is a general job shop.) In each cycle the system completes one unit of each product. Note that a single unit may require multiple cycles to complete all tasks. A unit may be one part or a batch of multiple parts that must be processed together.

A schedule defines the length of the cycle and the start time (within a cycle) of each task. The schedule is stable: the (relative) start times within each and every cycle are the same. That is, the interval between consecutive start times of each task remains the same (see, for example, Figure 1).

Careful scheduling is required to coordinate activities so that the system completes units at the highest possible throughput (frequency) and with the minimal amount of work-in-process inventory (WIP). Unfortunately, these two objectives conflict. Note that maximizing throughput is the same as minimizing the cycle length; many authors call this length the cycle time. To avoid confusion, this paper uses the term *cycle length*. In addition, we use the term *flow time*

(not cycle time) to describe the time that a unit spends in the production system between the start of its first task and the completion of its last task.

This paper will discuss the tradeoffs that exist between these two objectives. The paper presents approaches for maximizing throughput, minimizing WIP, and finding compromise solutions. In addition to various heuristics, the paper presents a novel problem space search. The paper discusses the results of experiments conducted to evaluate the performance of these approaches.

A number of papers have studied cyclic production systems that have limited internal storage and use a robot or other mechanism to move parts. Venkatesh *et al.* (1997) and Srinivasan (1998) analyze the performance of cluster tools. Jeng *et al.* (1993), Hall *et al.* (1997), and Crama and van de Klundert (1997) study the problem of sequencing activities in robot-centered cells.

In general, a cyclic production system can be modeled as an event graph when the schedule is given a priori, as shown in Hillion and Proth (1989), Proth and Xie (1996), and Minis and Proth (1995). It is then possible to derive the minimal WIP using the properties of the event graphs. Laftit *et al.* (1992) proposed several algorithms to reach a given cycle length while minimizing a linear combination of the markings, the coefficients of which are the components of a p-invariant.

Matsuo (1990) discussed the complexity of the cyclic scheduling problem in a two-machine flow shop, where the objective function is either to minimize the total idle time on the first machine or to minimize the total waiting time of the jobs processed in each cycle. Minimizing the total waiting time subject to the constraint that the cycle time is minimal is strongly NP-hard. Roundy (1992) analyzed the cyclic scheduling of a single product that revisits

some resources in its processing sequence. In this setting, minimizing the flow time subject to an upper bound on the cycle time (cycle length) is strongly NP-hard. Kamoun and Sriskandarajah (1993) discussed a number of flow shop, open shop, and job shop cyclic scheduling problems. Minimizing the cycle time (cycle length) in a two-machine no-wait job shop is strongly NP-hard, even if each job has only two tasks. Hall and Sriskandarajah (1996) reviewed a large number of complexity results for no-wait scheduling problems. Lee and Posner (1997) studied the problem of scheduling a minimal part set that needs to be produced a finite number of times when the sequence of tasks for each resource is given. Each resource must complete all required tasks in one MPS before beginning the next MPS (though some resources may be working on a previous or subsequent MPS).

Section 2 formulates the cyclic production scheduling problem and discusses the objective functions. Section 3 presents an approach for maximizing throughput. Section 4 presents an approach for reducing WIP. Section 5 presents a problem space search algorithm that generates a sequence of solutions that describe the tradeoff between WIP and throughput. Section 6 discusses experimental results. Section 7 concludes the paper.

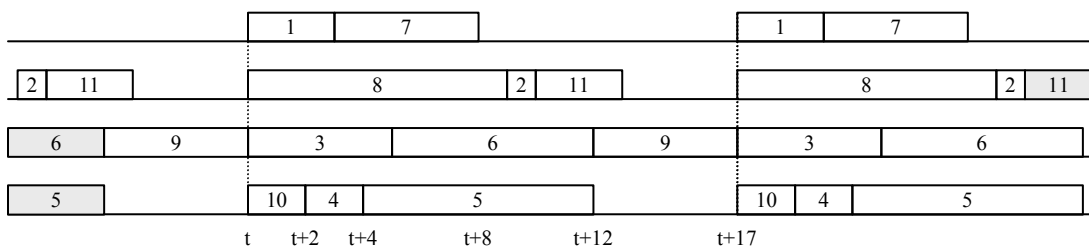


Figure 1. A feasible cyclic schedule.
(Shaded tasks are shown only in part.)

2. Problem Formulation

After describing the notation, this section will define the problem, give an example, and discuss the tradeoffs between throughput and WIP.

2.1 Notation

The cyclic production system has R resources and produces n different products. The system produces one unit of each product each cycle. A unit may be one part or a batch of multiple parts that must be processed together (no lot splitting is allowed). To produce these units, the system must perform e different tasks each cycle.

Let $r = 1, \dots, R$ index the resources. Let $b = 1, \dots, n$ index the products. Let $i = 1, \dots, e$ index the tasks. Let $r(i)$ be the resource that performs task i , which requires $d(i)$ time units. Note $R \leq e$ and $n \leq e$. Let $S(b)$ be the sequence of tasks required to produce one unit of product b . Let $q(b)$ be the number of tasks in $S(b)$. Note $q(1) + \dots + q(n) = e$. Let $E(b)$ be the set of immediate precedence constraints. If (i, j) is in $E(b)$, then task j follows task i in $S(b)$. Then, $j = \text{succ}(i)$, and $i = \text{pred}(j)$. Let $D(b)$ be the sum of the $d(i)$ for all tasks i in $S(b)$.

$$D(b) = \sum_{i \text{ in } S(b)} d(i).$$

Let $C(r)$ be the total processing requirements on resource r :

$$C(r) = \sum_{r(i)=r} d(i).$$

Let C^* be the maximum processing requirements: $C^* = \max \{C(1), \dots, C(R)\}$. Note that $D(1) + \dots + D(n) = C(1) + \dots + C(R)$.

2.2 Schedule Feasibility and Performance

Given the set of tasks and resources, a feasible cyclic schedule specifies the start time $x(i)$ and end time $y(i)$ of each required task i , $i = 1, \dots, e$, and a length C that satisfies the following

constraints for all i : $0 \leq x(i) < C$. $y(i) = x(i) + d(i)$. If $y(i) \leq C$, then resource $r(i)$ is executing task i in the interval $[x(i), y(i)]$. Otherwise, resource $r(i)$ is executing task i in the interval $[x(i), C]$ and in the interval $[0, y(i)-C]$. In either case, resource $r(i)$ is executing no other task when it is executing task i . (Note that we use the interval $[0, C]$ for convenience, since the cycle is continuously repeated and any interval of length C would be sufficient for defining the cyclic schedule.)

The relevant system performance measures are system throughput, average product flow time, and average work-in-process inventory (WIP). The system throughput equals the number of parts produced per unit time. Since a cycle of length C produces n units, the throughput $TH = n/C$. For a feasible cyclic schedule with length C , we measure WIP by first calculating the average product flow time and then employing Little's Law.

WIP Measurement.

- Step 1. For each product $b = 1, \dots, n$, perform Step 1a for the first task in $S(b)$ and perform Step 1b for each remaining task. Then go to Step 2.
- Step 1a. Let i be the first task in $S(b)$. Assign the label $w(i) = 0$ to this task.
- Step 1b. Let i be the next unlabeled task in $S(b)$ and let $j = \text{pred}(i)$. Label task i as follows:
If $y(j) > x(i)$, then let $w(i) = w(j) + 1$. Otherwise, let $w(i) = w(j)$.
- Step 2. For each product $b = 1, \dots, n$, let g be the first task in $S(b)$ and let h be the last task in $S(b)$. Then, the product flow time $Q(b) = w(h)C + y(h) - (w(g)C + x(g))$.
Since $w(g) = 0$ by definition, $Q(b) = w(h)C + y(h) - x(g)$.
- Step 3. The average product flow time is $(Q(1) + \dots + Q(n))/n$.
The average WIP is $(Q(1) + \dots + Q(n))/C$ units.

2.3 Example

Consider a cyclic production system that has $R = 4$ resources and produces $n = 3$ products. There are $e = 11$ different tasks that need to be performed. Table 1 lists the tasks (in sequence for each product), the resources that perform the tasks, and the task durations. Then, $C(1) = d(1) + d(7) = 3 + 5 = 8$. Similarly, $C(2) = 13$, $C(3) = 17$, and $C(4) = 12$. $C^* = C(3) = 17$.

Table 1. Example 1: Task List

Product b	Task i	Resource $r(i)$	Duration $d(i)$
1	1	1	3
	2	2	1
	3	3	5
	4	4	2
2	5	4	8
	6	3	7
	7	1	5
	8	2	9
3	9	3	5
	10	4	2
	11	2	3

Table 2 lists the start and end times of each task on each machine in a feasible cyclic schedule of length $C = 17$. See also Figure 1, which shows the cycle that starts at time t and portions of the adjacent cycles. In addition, Table 2 lists the labels that the WIP measurement procedure applies to the tasks. From this information, one can see that $Q(1) = 2C+4-0 = 38$. $Q(2) = 3C+9-4 = 56$, and $Q(3) = C+13-12 = 18$. The average product flow time equals $37 \frac{1}{3}$. The average WIP equals $6 \frac{10}{17}$.

Table 2. Example 1: A Feasible Cyclic Schedule.

Resource r	Task i	Start $x(i)$	End $y(i)$	Label $w(i)$
1	1	0	3	0
	7	3	8	2
2	8	0	9	3
	2	9	10	0
	11	10	13	1
3	3	0	5	1
	6	5	12	1
	9	12	17	0
4	10	0	2	1
	4	2	4	2
	5	4	12	0

2.4 Tradeoffs between Throughput and WIP

Theorem 1. *In any feasible cyclic schedule, the cycle length must be at least C^* , and the throughput cannot be greater than n/C^* , where $C^* = \max \{C(1), \dots, C(R)\}$.*

Proof. In each cycle, each resource r must complete all tasks i such that $r(i) = r$. Thus, the length of a cycle must be no less than $C(r)$. Thus, the length C is no less than $C^* = \max \{C(1), \dots, C(R)\}$. Since the throughput equals n/C , this must be no greater than n/C^* . ■

Let $TH^* = n/C^*$ denote the maximal throughput. Because the flow time of a unit must be at least the total processing time, $Q(b) \geq D(b)$ for all b . Therefore, the average product flow time must be greater than or equal to $F^* = (D(1) + \dots + D(n))/n$. Therefore, a feasible cyclic schedule that achieves the maximal throughput of n/C^* must yield a WIP greater than or equal to $W^* = F^* TH^* = (D(1) + \dots + D(n))/C^*$.

In general, a feasible cyclic schedule that achieves a throughput of TH must have a WIP greater than or equal to $F^* TH$. Thus, the point (WIP, TH) for any feasible cyclic schedule will lie on or to the left of the line $W^* = F^* TH$ and on or below the line $TH = n/C^*$, as shown in

Figure 2. In the set of all feasible schedules there are non-dominated solutions, which will form a tradeoff curve between WIP and throughput like that shown in Figure 2.

Although optimizing either one of the objectives is straightforward, combining the objectives yields a much more difficult problem, as discussed by Matsuo (1990), Roundy (1992), and Kamoun and Sriskandarajah (1993). Based on their results, it is clear that achieving both minimal WIP (where no unit waits) and maximal throughput is strongly NP-hard. However, Section 4.2 presents a special case where maximizing throughput and minimizing WIP is easy.

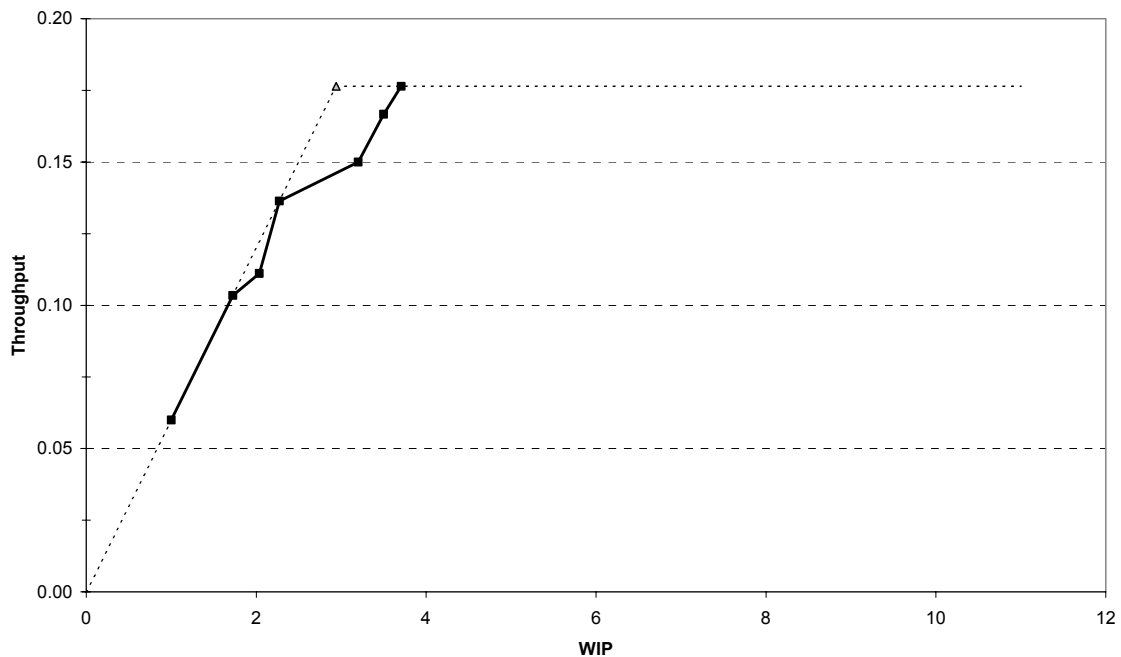


Figure 2. Example 1: Throughput versus WIP.

3. Achieving Maximal Throughput

Maximizing throughput (with no constraints on flow time or WIP) requires minimizing the length of each cycle. As discussed before, C^* is the minimal feasible cycle length, and the maximal throughput $TH^* = n/C^*$. This section presents a polynomial-time algorithm to create a feasible cyclic schedule that achieves the maximal throughput.

The simplest way to generate a schedule that achieves the minimal cycle length (and maximal throughput) is to ignore all of the precedence constraints. Given the set of tasks and resources, the *list scheduling algorithm* proceeds as follows:

Step 1. For each $r = 1, \dots, R$, let $A(r) = 0$.

Step 2. For each $i = 1, \dots, e$, let $x(i) = A(r(i))$ and $y(i) = x(i) + d(i)$. Increase $A(r(i))$ to $y(i)$.

For each resource r , this procedure completes all tasks in the interval $[0, C(r)]$. Thus, the cycle length is C^* . One can use the WIP Measurement procedure (Section 2.2) to calculate the average product flow time and average WIP of the schedule.

Theorem 2. *The list scheduling algorithm maximizes throughput in $O(e)$ time.*

Proof. The list scheduling algorithm constructs a feasible cyclic schedule that has a cycle length of C^* . Thus, this schedule achieves the maximal throughput. The algorithm requires $O(e)$ time since it simply calculates the start time each and every task. ■

Theorem 3. *The WIP of the cyclic schedule created by the list scheduling algorithm is bounded above by e .*

Proof. Each unit completes at least one task every cycle. Thus, a unit of product b requires at most $q(b)$ cycles, which have a length of C^* . Therefore, $Q(b)$ is no greater than $q(b)C^*$ for $b = 1, \dots, n$, and the average product flow time is no greater than $(q(1) + \dots + q(n))C^*/n = eC^*/n$. The throughput $TH = n/C^*$, so the WIP is no greater than e . ■

For Example 1, the list scheduling algorithm constructs the schedule presented in Table 3. The average product flow time equals 49, the throughput equals $3/17$, and the WIP equals $8 \frac{11}{17}$. Note that this is less than e , which equals 11.

Table 3. Example 1: The List Scheduling Schedule.

Resource r	Task i	Start $x(i)$	End $y(i)$
1	1	0	3
	7	3	8
2	2	0	1
	8	1	10
	11	10	13
3	3	0	5
	6	5	12
	9	12	17
4	4	0	2
	5	2	10
	10	10	12

4. Minimizing WIP

In the cyclic production system, the minimal reasonable WIP equals one. Any lower value would require the system to be completely idle for some period each cycle. This section presents an algorithm that constructs a feasible cyclic schedule with minimal WIP and algorithms that find compromise solutions.

4.1. The No-Wait Scheduling Algorithm

One can achieve the minimal WIP by creating a schedule in which no unit ever waits, and there is no idle time between successive units. Given the set of tasks and resources, the *no-wait scheduling algorithm* proceeds as follows:

- Step 1. Let $t = 0$. For each $b = 1, \dots, n$, perform Step 1a for the first task in $S(b)$ and perform Step 1b for each remaining task.
- Step 1a. Let i be the first task in $S(b)$. Let $x(i) = t$ and $y(i) = x(i) + d(i)$. Increase t to $y(i)$.
- Step 1b. Let i be the next task in $S(b)$ and let $j = \text{pred}(i)$. Let $x(i) = y(j) = t$ and $y(i) = x(i) + d(i)$. Increase t to $y(i)$.

Theorem 4. *The WIP of the cyclic schedule created by the no-wait scheduling algorithm equals 1.*

Proof. The cycle length $C = D(1) + \dots + D(n)$. The throughput $TH = n/C$. The flow time of one unit of product b is $D(b)$. The average product flow time is $(D(1) + \dots + D(n))/n = C/n$. By Little's Law, the $WIP = 1$. ■

Of course, one could further reduce the throughput and WIP by inserting idle time between the completion of the last task for any product and the start of the first task for the next product. In this way, the cycle length can be increased without bound, but the average product flow time remains the same. Thus, the throughput and the WIP can be made arbitrarily small.

For Example 1, the no-wait scheduling algorithm creates a feasible cyclic schedule with length $C = 50$. The throughput equals $3/50$, the average product flow time equals $50/3$, and the WIP equals 1.

4.2. The Shop Scheduling Algorithm

This section presents a heuristic to find a feasible cyclic schedule that completes all tasks for each unit within one cycle. Recall that no task sequences are given for any resource.

The *shop scheduling algorithm* proceeds as follows:

- Step 1. Let V be the set of completed tasks. $V = \{\}$. Let W be the set of remaining tasks. $W = \{1, \dots, e\}$. Let U be the set of ready tasks. Set $U = \{\}$. For each product $b = 1, \dots, n$, let j be the first task in $S(b)$, set $z(j) = 0$, and add task j to U .
- Step 2. Let $A(r)$ denote the available time of resource r . Set $A(r) = 0$ for all $r = 1, \dots, R$.
- Step 3. For each j in U , determine the earliest start time $x(j)$ as follows:

$$x(j) = \max\{z(j), A(r(j))\}.$$

- Step 4. Let $x^* = \min \{x(j) : j \text{ in } U\}$. If there exist j and k in U and a resource r such that $x(j) = x(k) = x^*$ and $r(j) = r(k) = r$, then go to Step 5. Otherwise, pick any j in U such that $x(j) = x^*$ and go to Step 6.
- Step 5. Let $U^*(r)$ be the tasks j in U such that $x(j) = x^*$ and $r(j) = r$. Using some rule, pick a task from $U^*(r)$.
- Step 6. Let $y(j) = x(j) + d(j)$. Increase $A(r(j))$ to $y(j)$. Remove j from W , remove j from U , and add j to V . If $\text{succ}(j)$ exists, let $z(\text{succ}(j)) = y(j)$ and add $\text{succ}(j)$ to U . If U is empty, stop. Otherwise, go to Step 3.

The cycle length $C = \max \{y(1), \dots, y(e)\}$. Next we present two rules to be used in Step 5: the most work remaining (MWR) rule and the least work remaining (LWR) rule. Let us define for each task i the quantity $\text{wr}(i)$ (the work remaining) as follows: $\text{wr}(i) = d(i) + \text{wr}(\text{succ}(i))$ if $\text{succ}(i)$ exists; $\text{wr}(i) = d(i)$ if $\text{succ}(i)$ does not exist.

MWR: Select one of the tasks that has the maximal value of $\text{wr}(j)$.

LWR: Select one of the tasks that has the minimal value of $\text{wr}(j)$.

The shop scheduling algorithm has to schedule e tasks, and the set U has at most n tasks in it, so the shop scheduling algorithm requires $O(ne)$ effort.

Theorem 5. *The WIP of the cyclic schedule created by the shop scheduling algorithm is no greater than n .*

Proof. The cycle length $C = \max \{y(1), \dots, y(e)\}$. The throughput $\text{TH} = n/C$. For any task i , if task $i = \text{pred}(j)$ ($j = \text{succ}(i)$), $z(j) = y(i)$, the completion time of task i (Step 6). By Step 3, $x(j) \geq z(j) = y(i)$, and $y(j) \leq C$, so task i and task j are done in the same cycle (that is, the WIP measurement algorithm would set $w(j) = w(i)$). This is true for all tasks (all tasks have the label

$w(i) = 0$). The product flow time $Q(b) \leq C$ for all $b = 1, \dots, n$. Thus, the average product flow time is less than or equal to C , and the WIP is less than or equal to n by Little's Law. ■

Theorem 6. *If $q(b) = 1$ for all $b = 1, \dots, n$, then the shop scheduling algorithm creates a feasible cyclic schedule with length = C^* , throughput = TH^* , and WIP = W^* .*

Proof. Recall that $C^* = \max\{C(1), \dots, C(R)\}$, $TH^* = n/C^*$, and $W^* = (D(1) + \dots + D(n))/C^*$.

The cycle length $C = \max\{y(1), \dots, y(e)\}$. Because $q(b) = 1$ for all $b, e = n$. Since every task is the first task in some $S(b)$, $z(j) = 0$ for $j = 1, \dots, e$. Thus, $x(j) = A(r(j))$ for all j , but $A(r(j)) = y(i)$ if task i was the last task scheduled on the resource. $A(r(j)) = 0$ if there was no previously scheduled task. Thus, there will be no idle time between consecutive tasks on the same resource.

The maximal task completion time must be C^* , so $C = C^*$ and $TH = n/C^*$. Since each product has just one task, $Q(b) = D(b)$ for all b . Thus, the average product flow time equals $F^* = (D(1) + \dots + D(n))/n$, and the WIP equals $(D(1) + \dots + D(n))/C^* = W^*$ by Little's Law. ■

For Example 1, the shop scheduling algorithm (with either rule) generates the schedule shown in Table 4. See also Figure 3. The cycle length $C = 31$. $Q(1) = 12$, $Q(2) = 31$, and $Q(3) = 13$. The throughput equals $3/31$, the average product flow time is $18 \frac{2}{3}$, and the WIP equals $1 \frac{25}{31}$.

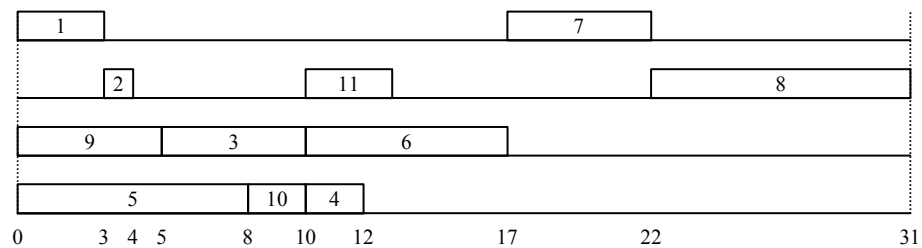


Figure 3. Example 1: Schedule constructed by the shop scheduling algorithm.

Table 4. Example 1: A feasible cyclic schedule generated by the shop scheduling algorithm.

Resource r	Task i	Start $x(i)$	End $y(i)$
1	1	0	3
	7	17	22
2	2	3	4
	11	10	13
	8	22	31
3	9	0	5
	3	5	10
	6	10	17
4	5	0	8
	10	8	10
	4	10	12

4.3 The MPS Scheduling Algorithm

As discussed previously, the approach of Lee and Posner (1997) constructs a feasible cyclic schedule with minimal cycle length if, for each resource, a sequence of the required tasks is given. These sequences, along with the sequences that define each product's routing, form a directed graph that must contain no cycles. Thus, the Lee and Posner approach can reduce the cycle length (and increase the throughput) of schedules generated by the shop scheduling algorithm. For Example 1, given the sequences in Table 4, Tasks 1 and 2 are delayed one time unit. In the new cyclic schedule, the cycle length $C = 27$. $Q(1) = 11$, $Q(2) = 31$, and $Q(3) = 13$. The throughput equals $3/27 = 1/9$, the average product flow time is $18 \frac{1}{3}$, and the WIP equals $2 \frac{1}{27}$. Compared to the schedule in Table 4, this schedule has higher throughput but also larger WIP.

The *MPS scheduling algorithm* proceeds as follows:

Step 0. Generate an initial schedule using the shop scheduling algorithm. Let E be the set of precedence constraints between tasks defined by the machine sequences (from the

schedule) and the technology constraints (from the $E(b)$). Let V equal the set of nodes that correspond to the tasks. The length of any edge (i, j) in E equals $d(i)$, the processing time of task i . Let $ft(r)$ be the first task on resource r . Let $lt(r)$ be the last task on resource r . Let $M = \{ft(r) \text{ for all } r\}$. If $j = ft(r)$, let $p(j) = lt(r)$, and $f(j) = d(lt(r))$, the processing time of task $lt(r)$. Thus, $p(j)$ and $f(j)$ are defined for all j in M . Let $VV = V + \{u\}$, where u is a dummy start node for the problem. Let $EE = E + \{(p(j), j) \text{ for all } j \text{ in } M\} + \{(u, j) \text{ for all } j \text{ in } M\}$. (See Figure 4.)

- Step 1. Calculate $Q(i, j)$, the length of the longest path from i in V to j in V using arcs in E . Note that E is acyclic.
- Step 2. Let $g_0(w, v) = Q(w, v)$ for all w in M and v in V .
- Step 3. For $k = 1, \dots, R$ (the number of resources) perform Steps 4 and 5.
- Step 4. Let $g'_k(w, j) = g_{k-1}(w, p(j)) + f(j)$ for all w in M and j in M . This represents the longest path (in EE) from w to j that has exactly k reverse edges and ends with a reverse edge (from $p(j)$ to j).
- Step 5. Let $g_k(w, v) = \max\{g'_k(w, j) + Q(j, v) \text{ over } j \text{ in } M\}$ for all w in M and v in V . This represents the longest path (in EE) from w to v that has exactly k reverse edges.
- Step 6. For $k = 1, \dots, R$, let $\lambda_k = (1/k) \max\{g_k(w, w) \text{ over } w \text{ in } M\}$.
Let $\lambda^* = \max\{\lambda_1, \dots, \lambda_R\}$.
- Step 7. For all j in M , let the length of edge $(p(j), j)$ equal $f(j) - \lambda^*$ and the length of edge (u, j) equal 0. Calculate $H(u, v)$ the length of the longest path from u (the dummy start node) to v in V using arcs in EE . Note that no cycle has a positive length due to the definition of λ^* . Each task v starts at time $x(v) = H(u, v)$, and the cycle length $C = \lambda^*$.

The above algorithm uses the algorithms of Lee and Posner (1997) and Karp and Orlin (1981). In Step 0, the shop scheduling algorithm requires $O(ne)$ effort. Due to the special structure of (V, E) , Step 1 can be done in $O(e^3)$ time. Since M has R elements and V has e elements, Step 2 requires $O(Re)$ effort. Step 4 requires $O(R^2)$ time. Step 5 requires $O(R^2e)$ time. Together, Steps 3, 4, and 5 require $O(R^3e)$ effort. Step 6 requires $O(R^2)$ time. For Step 7, we use the FIFO label-correcting algorithm presented by Ahuja *et al.* (1993, page 142). The number of nodes in E equals the number of tasks e , and the number of edges in V is bounded by $2e$, since each task has at most two precedence constraints (one for the product sequence, and one for the resource sequence). Thus, the number of nodes in EE equals $e + 1$, and the number of edges in VV is bounded by $2e + 2r \leq 4e$. Thus, the shortest path algorithm runs in $O(e^2)$ time. Since $n \leq e$, the total effort of the MPS scheduling algorithm is $O(R^3e + e^3)$.

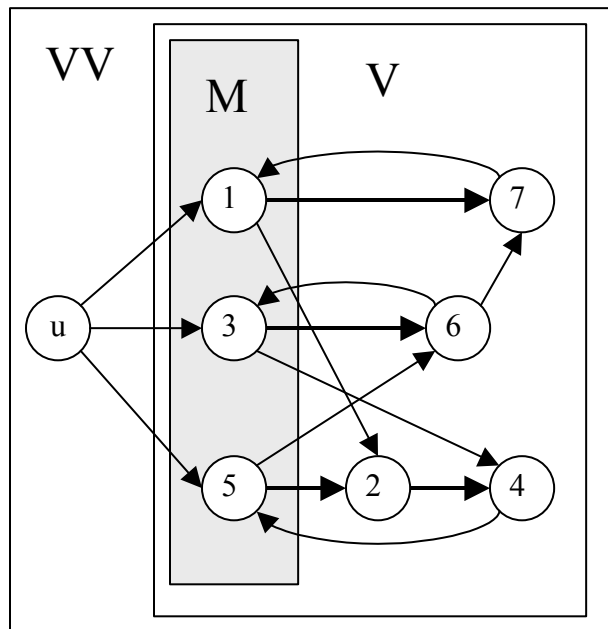


Figure 4. Initialization of MPS Scheduling Algorithm.

5. A Tradeoff Exploration Search in Problem Space

This section presents a search algorithm that explores the tradeoffs between WIP and throughput. The search explores a problem space similar to those considered by Storer *et al.* (1992) and Herrmann and Lee (1995). Searching the problem space exploits a heuristic's ability to find good solutions based on the structure of the problem but allows exploration to find even better solutions. The search algorithm finds a sequence of feasible cyclic schedules. The goal is to provide a set of schedules that have different values of WIP and throughput so that a decision-maker can select the most appropriate one.

The initial solution is the solution that the shop scheduling algorithm generates from the problem instance. At each iteration, the search algorithm creates a number of new instances, uses the shop scheduling algorithm to find a solution for each, and selects the instance that yielded the shortest cycle length. To create a new instance, it removes the precedence constraint on a task along a critical path. That is, the algorithm creates a new trial instance by splitting a product sequence and adding a new product for the second part of the sequence. Eventually, the final solution will be a maximal throughput solution, which must happen when all of the precedence constraints are removed, according to Theorem 6.

The *tradeoff exploration search* proceeds as follows:

Step 1. Let $P(0)$ be the original problem instance. Let $n(0) = n$ products. Use the shop scheduling algorithm (with the MWR rule) to find a feasible cyclic schedule. Let $C(0)$ be the resulting schedule length. Output the schedule created for $P(0)$. If $C(0) = C^*$, then stop. Otherwise, let $k = 0$.

- Step 2. Determine the earliest and latest start times of each task for the current set of $n(k)$ products (assuming that all tasks are available at time 0 and must be done by time $C(k)$). For each task $i = 1, \dots, e$, perform Step 2a.
- Step 2a. If the earliest start time of task i equals the latest start time of task i , then task i is on a critical path for the current solution, and go to Step 2b. Otherwise, let $L(i) = M$ (some large number greater than $C(0)$) and skip Step 2b.
- Step 2b. Create a new product, numbered $n(k)+1$. Let b be the product that requires task i . Remove from $S(b)$ the subsequence that starts with task i and ends with the last task in $S(b)$. Let this removed subsequence be $S(n(k)+1)$, the sequence of tasks for the new product. Call the modified instance $I(i)$. Use the shop scheduling algorithm to find a feasible cyclic schedule. Let $L(i)$ be the resulting schedule length.
- Step 3. Select instance $I(i)$ with the minimum $L(i)$ (breaking ties randomly). Let $P(k+1) = I(i)$, $n(k+1) = n(k)+1$, $C(k+1) = L(i)$. Output the schedule created for $P(k+1)$. If $C(k+1) = C^*$, then stop. Otherwise, let $k = k+1$ and return to Step 2.

Recall that Theorem 6 shows that if the number of products equals the number of tasks, the shop scheduling algorithm will yield a schedule with cycle length equal to C^* . Thus, Step 2 is entered at most $e-n$ times. Each execution of Step 2 requires at most $e-n$ runs of the shop scheduling algorithm. And the shop scheduling algorithm is $O(ne)$. Thus, the total effort is $O((e-n)^2 ne)$.

6. Algorithm Performance

Experimental testing evaluated six scheduling algorithms and the tradeoff exploration search on sixteen sets of problem instances. The primary goal of these experiments was to determine the average quality of the schedules that each algorithm constructed. In this case,

quality is determined by the throughput and WIP of the schedule. Since the computational complexity of each algorithm is known, testing did not consider the objective of computational effort. The following seven algorithms were tested:

1. The no-wait scheduling algorithm (NWS)
2. The shop scheduling algorithm with MWR rule (SS-MWR)
3. The shop scheduling algorithm with LWR rule (SS-LWR)
4. The MPS scheduling algorithm with the MWR rule (MPS-MWR)
5. The MPS scheduling algorithm with the LWR rule (MPS-LWR)
6. The list scheduling algorithm (LS)
7. The tradeoff exploration search

6.1 Instance Generation

The systematic generation of problem instances considered the following attributes: number of products, number of tasks per product, number of resources, variation in processing times between tasks on the same resource and variation of means among each resource. There were four cases (which corresponded to the size of the problem) and, within each case, four variations (which corresponded to different levels of variability). Altogether, we generated 16 problem sets, each with 10 instances. See Tables 5 and 6 for complete lists of the attribute values.

Table 5. Parameter values for Cases PA, PB, PC, and PD.

Case	PA	PB	PC	PD
Number of resources R	5	5	25	25
Number of products n	5	25	5	25
Tasks per product v	5	5	25	5
Total number of tasks e	25	125	125	125

Within each case, two of the four variations (00, 01) used the same processing time distribution on every resource. Instances in the other two variations (10, 11) had five different

processing time distributions. Variations 00 and 10 used uniform distributions, while variations 01 and 11 used geometric distributions (which have higher variability). In cases PA and PB, each distribution was assigned to one resource. In cases PC and PD, each distribution was assigned to five resources. In all problem sets, a resource was randomly assigned to each task, but the sequence of tasks for a product did not visit any resource more than once. (That is, there was no re-entrant flow.) The randomly generated processing times were integers. The variability of the uniform random variables (measured by their squared coefficient of variation) ranges from 0.128 to 0.16. The variability of the geometric random variables ranges from 0.8 to 0.96.

Table 6. Probability distributions for Variations 00, 01, 10, and 11.

Variation	00	01	10	11
Mean processing time	15	15	5, 10, 15, 20, 25	5, 10, 15, 20, 25
Processing time distribution(s)	Uniform [6, 24]	Geometric $p = 1/15$	Uniform [2, 8], [4, 16], [6, 24], [8, 32], [10, 40]	Geometric $p = 1/5$, $p = 1/10$, $p = 1/15$, $p = 1/20$, $p = 1/25$

Given n , R , and $v = e/n$, the instance generation algorithm proceeded as follows. Note that, in Step 3, removing $r(i)$ from $A(b)$ ensures that each task in $S(b)$ requires a different resource.

- Step 1. For each product $b = 1, \dots, n$, let $A(b) = \{1, \dots, R\}$ and perform Step 2.
- Step 2. $S(b) = \{(b-1)v + 1, \dots, bv\}$. For each task i in $S(b)$, perform Step 3.
- Step 3. Randomly select a resource $r(i)$ in $A(b)$ (all are equally likely). Remove $r(i)$ from $A(b)$. Randomly select a processing time $d(i)$ from the processing time distribution for $r(i)$.

6.2 Experimental Results

Each scheduling algorithm was used to construct a solution to each instance. The tradeoff exploration search was used to find a sequence of solutions to each instance. For each instance, C^* , TH^* , and W^* were calculated and used as benchmarks for reporting algorithm performance. That is, the throughput and WIP of each cyclic schedule was normalized by dividing it by the TH^* and W^* for that instance. The presentation of the results below uses these normalized measures, which are called relative WIP and relative throughput.

The primary result was that the performance of the scheduling algorithms varied greatly depending on the case. The variability of processing times did not affect the performance as greatly. Therefore, all reported results are averaged over the four variations. Table 7 lists the performance of each heuristic on the forty instances in all four variations of each case. Note that AR WIP means the average relative WIP, and AR TH means the average relative throughput.

In cases PA and PD, the shop scheduling algorithm and MPS scheduling algorithm (SS-LWR, MPS-LWR, SS-MWR, and MPS-MWR) constructed solutions that had high relative throughput (0.74 to 0.94) and moderate relative WIP (1.05 to 1.54). In case PB, these algorithms constructed solutions that had higher relative throughput (0.85 to 0.99) and much larger relative WIP (1.94 to 4.49). In this case, the MWR rule increased WIP and throughput significantly. In case PC, these algorithms constructed solutions that had low relative throughput (0.29 to 0.33) and low relative WIP (0.33 to 0.37). In all cases, the no-wait scheduling algorithm constructed schedules that had much lower relative WIP and relative throughput, while the list scheduling algorithm constructed schedules with maximal throughput but excessive relative WIP.

Using the MPS algorithm to improve the schedules generated by the shop scheduling algorithm increases the schedules' WIP and throughput. On average, the influence is greater in cases PA and PD than in cases PB and PC. See Table 7 and Figure 5.

Table 7. Results for heuristics on all four cases.

Heuristic	PA		PB		PC		PD	
	AR WIP	AR TH	AR WIP	AR TH	AR WIP	AR TH	AR WIP	AR TH
NWS	0.32	0.32	0.28	0.28	0.08	0.08	0.09	0.09
SS-LWR	1.05	0.74	1.94	0.85	0.33	0.29	1.07	0.74
MPS-LWR	1.12	0.81	1.95	0.86	0.35	0.32	1.28	0.84
SS-MWR	1.23	0.78	4.47	0.99	0.35	0.31	1.39	0.87
MPS-MWR	1.34	0.87	4.49	0.99	0.37	0.33	1.54	0.94
LS	4.87	1.00	17.26	1.00	7.30	1.00	6.53	1.00

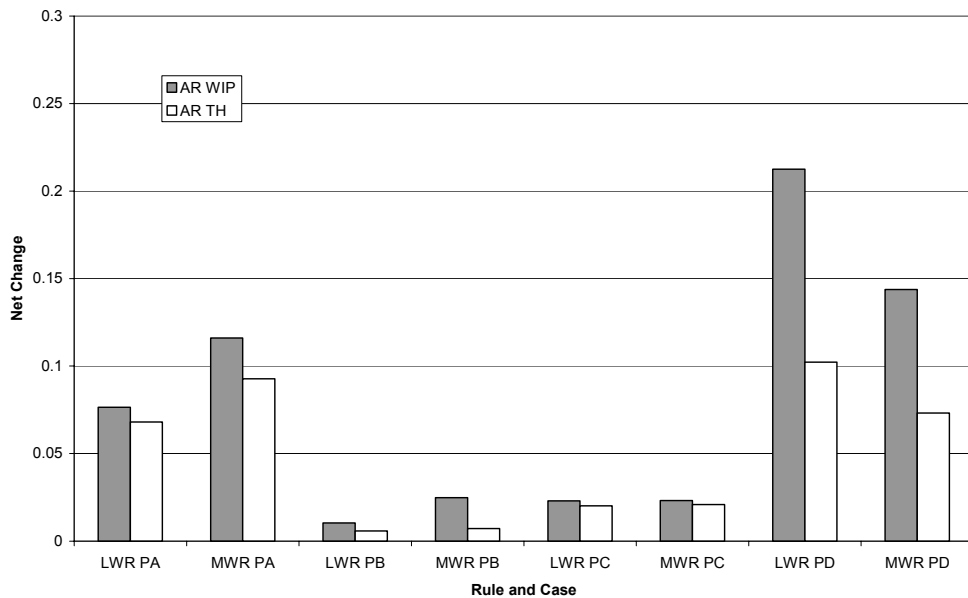


Figure 5. Net change due to MPS scheduling algorithm.

Figure 6 shows the results of the tradeoff exploration search for each case. (Note that the curves for cases PA and PD overlap.) The search's initial solution is the schedule generated by the shop scheduling algorithm (using the MWR rule). Thus, the performance of the search is influenced greatly by that heuristic's performance. In all cases, the search found solutions with higher throughput and higher WIP. In cases PA and PD, the average relative WIP of solutions with maximal throughput was 1.87 and 1.72. In case PB, the average relative WIP of solutions

with maximal throughput was 4.62. In case PC, the average relative WIP of solutions with maximal throughput was 2.24. Note that the search generated maximal throughput schedules that have much lower relative WIP than those generated by the list scheduling algorithm.

The results also show that, in some cases, the search converged to maximal throughput solutions quickly. In case PB, the search required just one iteration to reach the maximal throughput (averaged over 40 instances). In case PA, the search required six iterations; in case PD, the search required eight iterations. In case PC, however, the search started with very low relative throughput solutions, and the search required 43 iterations to reach the maximal throughput (averaged over 40 instances). In cases PA, PC, and PD, the search provides a series of solutions that allow a decision-maker to tradeoff the WIP and throughput objectives.

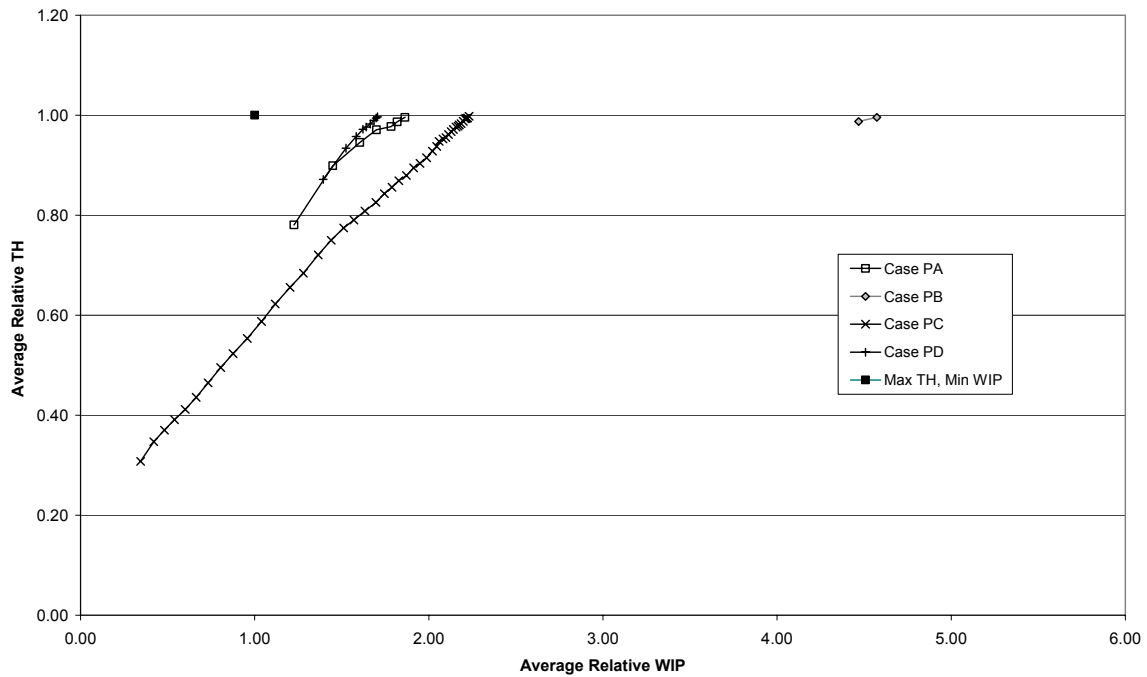


Figure 6. Performance of tradeoff exploration search.

6.3 Discussion

The structure of the cases helps explain the performance of the scheduling algorithms. In cases PA and PD, the number of products equals the number of resources, so products generally don't have to wait much, and the resources stay busy. In case PB, the number of products was much larger than the number of resources, so products have to wait a great deal (increasing WIP). In case PC, the number of products is small, but each product has many tasks. Thus, resources are often idle, and it is hard to achieve the maximal throughput, though the WIP is low.

One possible interpretation is to see the shop scheduling algorithm as an approach to maximize throughput under the constraint that the WIP is no greater than n . However, the relative sizes of n and W^* can vary greatly. Let p be the average duration of a task. Then, C^* is approximately $(e/R)p$ since there are e/R tasks on each resource. For all products $b = 1, \dots, n$, $D(b)$ is approximately vp , and the sum $D(1) + \dots + D(n)$ is approximately $nvp = ep$. Thus, W^* is approximately $ep/C^* = R$. In Cases PA and PD, $n = R$. However, in case PB, $n = 25$ and is much greater than R , and, in case PC, $n = 5$ and is much smaller than R .

Similarly, in Cases PA and PD, the shop scheduling algorithm constructed solutions with WIP near W^* , which is approximately n . In case PB, the shop scheduling algorithm constructed solutions with WIP much greater than W^* . In case PC, the shop scheduling algorithm constructed solutions with WIP much less than W^* .

The tradeoff exploration search performed well in case PC because it created new instances by dividing the long sequences (of 25 tasks) into shorter sequences. But in case PB, the sequences were already short so the search could not improve the solution quality greatly.

7. Summary and Conclusions

This paper presented the deterministic cyclic scheduling problem and discussed the tradeoffs between two important performance measures: WIP and throughput. Optimizing either objective function in isolation is extremely easy. However, usually some tradeoff between these two measures is necessary. Achieving maximal throughput requires excessive inventory. A no-wait cyclic schedule minimizes WIP but has very low throughput.

The paper does present a special case where maximizing throughput and minimizing WIP is easy. However, in general, it is a strongly NP-hard problem.

This paper presented algorithms for constructing cyclic schedules. The list scheduling algorithm maximizes throughput, while the no-wait scheduling algorithm minimizes WIP. The shop scheduling algorithm attempts a compromise by finding schedules with high throughput and reasonable WIP. The MPS scheduling algorithm adjusts these schedules to increase throughput. The tradeoff exploration search visits a series of points in the problem space to generate a series of schedules that converge in a solution with maximal throughput.

An experimental study provided results for evaluating the solution quality of the proposed algorithms. The study used 160 randomly generated problem instances that ranged in problem size and in the amount of processing time variability. Processing time variability had less influence on solution quality than problem size.

In general, the scheduling heuristics generated high throughput, moderate WIP solutions. The MPS algorithm increased WIP and throughput. The no-wait schedules had much lower relative WIP and relative throughput, while the list scheduling algorithm constructed schedules with maximal throughput but excessive relative WIP. The tradeoff exploration search provides a series of solutions that allow a decision-maker to tradeoff the WIP and throughput objectives.

Certainly a better job shop scheduling algorithm (like the shifting bottleneck approach) could generate solutions with shorter cycle length and higher throughput than the shop scheduling and MPS scheduling algorithms, though the computational effort would be greater. The shop scheduling algorithm is ideal, however, for the problem space search, which must call the heuristic repeatedly.

The cyclic scheduling problem is an interesting scheduling problem because the solution (which is continually repeated) significantly affects the performance of a dynamic production system. However, it is not clear which schedule is “best,” since there exist multiple objectives. This paper seeks to explain the behavior of this problem and provide decision-makers insight and approaches that will help manage such systems effectively. However, more work remains to extend the analysis to other domains.

Bibliography

- Crama, Yves and Joris van de Klundert, "Cyclic scheduling of identical parts in a robotic cell," *Operations Research*, Volume 45, Number 6, pages 952-965, 1997.
- Hall, Nicholas G., Hichem Kamoun, and Chelliah Sriskandarajah, "Scheduling in robotic cells: classification, two and three machine cells," *Operations Research*, Volume 45, Number 3, pages 421-439, 1997.
- Hall, Nicholas G., and Chelliah Sriskandarajah, "A survey of machine scheduling problems with blocking and no-wait in process," *Operations Research*, Volume 44, Number 3, pages 510-525, 1996.
- Herrmann, Jeffrey W., and Chung-Yee Lee, "Solving a class scheduling problem with a genetic algorithm," *ORSA Journal on Computing*, Volume 7, Number 4, pages 443-452, 1995.
- Hillion H.P., Proth J.-M., "Performance Evaluation of Job-Shop Systems Using Timed Event-Graphs," *IEEE Transactions on Automatic Control*, Volume 34, Number 1, pages 3-9, January 1989.
- Jeng, Wu-De, James T. Lin, and Ue-Pyng Wen, "Algorithms for sequencing robot activities in a robot-centered parallel-processor workcell," *Computers in Operations Research*, Volume 20, Number 2, pages 185-197, 1993
- Kamoun, H., and C. Sriskandarajah, "The complexity of scheduling jobs in repetitive manufacturing systems," *European Journal of Operational Research*, Volume 70, Number 3, pages 350-364, 1993.
- Laftit, S., J.-M. Proth, and X.-L. Xie, "Optimization of Invariant Criteria for Event Graphs," *IEEE Transactions on Automatic Control*, Volume 37, Number 5, pages 547-555, May 1992.
- Lee, Tae-Eog, and Marc E. Posner, "Performance measures and schedules in periodic job shops," *Operations Research*, Volume 45, Number 1, pages 72-91, 1997.

- Matsuo, Hirofumi, "Cyclic scheduling problems in the two-machine permutation flow shop: complexity, worst-case, and average-case analysis," *Naval Research Logistics*, Volume 37, pages 679-694, 1990.
- Minis, I., and J.-M. Proth, "Production Management in A Petri Net Environment," *Recherche Opérationnelle*, Volume 29, Number 3, pages 321-352, 1995.
- Pinedo, Michael, *Scheduling: Theory, Algorithms, and Systems*, Prentice Hall, Englewood Cliffs, New Jersey, 1995.
- Proth, J.-M., and X.-L. Xie, *Petri Nets. A Tool for Design and Management of Manufacturing Systems*, John Wiley and Sons Ltd, UK, 1996.
- Roundy, Robin, "Cyclic schedules for job shops with identical jobs," *Mathematics of Operations Research*, Volume 17, Number 4, pages 842-865, 1992.
- Srinivasan, R. S., "Modeling and performance analysis of cluster tools using Petri nets," *IEEE Transactions on Semiconductor Manufacturing*, Volume 11, Number 3, pages 394-403, 1998.
- Storer, R.H., S.Y.D. Wu, and R. Vaccari, "New search spaces for sequencing problems with application to job shop scheduling," *Management Science*, Volume 38, Number 10, pages 1495-1509, 1992.
- Venkatesh, Srilakshmi, Rob Davenport, Pattie Foxhoven, and Jaim Nulman, "A steady-state throughput analysis of cluster tools: dual-blade versus single-blade robots," *IEEE Transactions on Semiconductor Manufacturing*, Volume 10, Number 4, pages 418-424, 1997.