Optimization of Cyclic Production Systems: A Heuristic Approach

Fabrice Chauvet, Jeffrey W. Herrmann, and Jean-Marie Proth

***Abstract*—In this paper, the expression "production systems" refers to flow shops, job shops, assembly systems, Kanban systems, and, in general, to any discrete event system which transforms raw material and/or components into products and/or components. Such a system is said to be cyclic if it provides the same sequence of products indefinitely. A schedule of a cyclic production system is defined as soon as the starting time of each operation on the related resource is known. It has been shown that, whatever the feasible schedule applied to the cyclic production system, it is always possible to fully utilize the bottleneck resource. In other words, it is always possible to maximize the throughput of such a system. As a consequence, we aim at finding the schedule which permits to maximize the throughput with a work in process as small as possible. We propose a heuristic approach based on Petri nets to find a near-optimal, if not optimal, solution. We also give a sufficient condition for a solution to be optimal.**

***Index Terms*—Cyclic production systems, event graphs, Petri nets, scheduling, work in process (WIP).**

I. 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 items each cycle. Careful scheduling is required to coordinate the resources so that the system produces items at the highest possible frequency and with the minimal amount of work-in-process (WIP) inventory.

This paper considers the scheduling of cyclic production systems and describes a new approach that uses Petri nets. The problem is to determine the time at which each task should begin during the cycle. We assume that the set of tasks does not change, the tasks are nonpreemptive, and the task processing times are deterministic. Readers are supposed to be familiar with Petri nets and event graphs, otherwise they can refer to [5]–[7] or [8] for more information.

In Section II, we show that it is always possible to fully utilize the bottleneck machine and achieve maximum throughput, although the WIP depends upon the particular schedule. A sufficient condition for a feasible solution to be optimal is proposed in Section III. Section IV presents a powerful heuristic algorithm. This algorithm generates a near-optimal solution. Section V concludes the paper.

II. Maximizing Throughput for a Given Schedule

In this section, we consider the objective of maximizing the throughput. This is equivalent to reducing the length of the cycle and utilizing the bottleneck machine fully. Two approaches are proposed.

The first one, called the elementary approach, assigns the tasks to the resources in an arbitrary order, without taking into account the WIP, and fixing the cycle time to the time required by the bottleneck machine to complete a cycle if it is fully utilized. The WIP is adjusted to reach this goal. We present this approach to show that it is always possible to fully utilize the bottleneck machine. However, it should be noticed that in this approach, tasks are scheduled on the resources independently of their order in the manufacturing process. This order is recreated by means of WIP. In other words, WIP is considered as an adjustment parameter and not as a quantity that should be minimized.

Any cyclic production system can be modeled as an event graph when the schedule is given *a priori*, as shown in [6]–[8]. It is then possible to derive the minimal WIP using the properties of the event graphs. We refer to this approach as the event graph approach. As in the elementary approach, this approach schedules tasks on the resources independently of their order in the manufacturing process. The difference is that the ordered tasks are forced to belong to the same manufacturing process, and the schedule is adjusted by locating the initial WIP adequately to reduce it as much as possible.

Before reminding the reader of these methods, we introduce a simple example, which will be used in this paper to illustrate the proposed approaches.

A. Notation

We start by introducing some additional notation that precisely formulates the problem.

Let $b = 1, \ldots, n$ be the index over the different products. (The cyclic production system produces one unit of each product each cycle.)

Let $i = 1, \ldots, e$ be the index over all of the different tasks that need to be performed during each cycle. Let $r(i)$ be the resource that performs task i.

Let $d(i)$ be the duration of task i.

Let $S(b)$ be the sequence of tasks required to produce one unit of product b. Let $q(b)$ be the number of tasks required to produce one unit of product b. (We assume that there is a simple sequence of tasks. One could easily extend our results to the case where some tasks can be done in parallel. In other words, an assembly system can also be modeled as an event graph.)

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, $i = \text{pred}(j)$, and $j = \text{succ}(i)$, where pred stands for "predecessor" and succ for "successor."

Let $r = 1, \ldots, R$ be the index over the resources, $x(i, r) = 1$ if $r(i) = r$, and 0, otherwise.

Then, let $C(r)$ be the total processing requirements on resource r: $C(r) = x(1, r)d(1) + \cdots + x(e, r)d(e)$.

Let C^* be the maximum processing requirements: $C^* = \max\{C(1), \ldots, C(R)\}$.

B. Example

For the sake of clarity and simplicity, we consider the job shop already presented in [5] that has $R = 4$ machines and produces $n = 4$ products, where products three and four have the same manufacturing process. There are $e = 13$ different tasks that need to be performed. Table I lists the tasks in sequence for each product, the resources that perform the tasks, and the task duration.

We then derive the total processing requirements on each resource (machine) during a cycle

$$C(1) = d(1) + d(5) + d(8) + d(11) = 1 + 1 + 1 + 1 = 4.$$

Manuscript received December 20, 2001; revised July 31, 2002. This paper was recommended for publication by Associate Editor M. Jeng and Editor N. Viswanadham upon evaluation of the reviewers' comments.

F. Chauvet is with Bouygues Telecom, 78 944 Velizy, France (e-mail: fchauvet@bouyguestelecom.fr).

J. W. Herrmann is with the Institute for Systems Research, University of Maryland at College Park, College Park, MD 20742 USA (e-mail: jwh2@Glue.umd.edu).

J.-M. Proth is with INRIA/SAGEP, Universite de Metz, 57012 Metz Cedex 01, France (e-mail: Jean-Marie.Proth@loria.fr).

Digital Object Identifier 10.1109/TRA.2002.807529

TABLE I
MANUFACTURING PROCESSES

Product	Task	Resource	Duration
1	1	1	1
	2	2	1
	3	3	3
	4	4	3
2	5	1	1
	6	4	1
	7	3	2
3	8	1	1
	9	2	2
	10	4	1
4	11	1	1
	12	2	2
	13	4	1

Similarly, $C(2) = 5$, $C(3) = 5$, and $C(4) = 6$. $C^* = C(4) = 6$, so the bottleneck machine is machine four, and the minimal cycle time is six time units. Thus, the maximal throughput is $4/6 = 2/3$ items per time unit.

C. Elementary Approach

We begin by presenting the algorithm. Afterwards, we will apply it to the example presented above. This algorithm determines a feasible schedule and how many cycles are needed to complete one item of each product, given that schedule.

Algorithm 1:

1) For each resource, schedule the tasks that require that resource in any order. Let $x(i)$ and $y(i)$ denote the start and end times of task i. All tasks must start in the interval $[0, C^*]$, and the resource can perform, at most, one task at a time. (Thus, if a task ends at time $\theta > C^*$, the resource can perform no other task during the interval $[0, \theta - C^*]$.) No tasks can be preempted.
2) For each product $b = 1, \ldots, n$, perform step 3) for the first task in $S(b)$ and perform step 4) for each remaining task.
3) Let i be the first task in $S(b)$. Assign the label $w(i) = 0$ to this task.
4) 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 set $w(i) = w(j)+1$. Otherwise, set $w(i) = w(j)$.
5) For each product $b = 1, \ldots, n$, let g be the first task in $S(b)$ and h be the last task in $S(b)$. Then, calculate the product cycle time as follows:

$$Q(b) = (w(h) - w(g))C^* + y(h) - x(g).$$

Since $w(g) = 0$ by definition

$$Q(b) = w(h)C^* + y(h) - x(g).$$

6) The total average WIP is $Q(1) + \cdots + Q(n)/C^*$ items.

Let us apply this algorithm to the example. Recall that $C^* = 6$. In step 1), we create the following schedule based on the sequencing of the products on the resources as given in [5]. Machine 1 performs task 1 and then tasks 5, 8, and 11. Machine 2 performs tasks 2, 9, and 12 successively. Machine 3 performs task 3 and then task 7. Machine 4 performs tasks 4, 6, 10, and finally 13.

Now, we repeat steps 2)–4) and label each task. Consider product 1. The production system must perform tasks 1, 2, 3, and 4 to complete one item of product 1. Given the above schedule, since task 2 starts before task 1 ends, a particular item cannot undergo both tasks in the same cycle (which we call cycle 0), and the item must undergo task 2 in the next cycle (cycle 1). Similarly, since task 3 starts before task 2 ends, the item must undergo task 3 in the next cycle (cycle 2). Likewise, since task 4 starts before task 3 ends, the item must undergo task 4 in the cycle after that (cycle 3). The total cycle time for that item $Q(1) = 3 * 6 + 3 - 0 = 21$ time units. Likewise, $Q(2) = 10$, and $Q(3) = Q(4) = 9$.

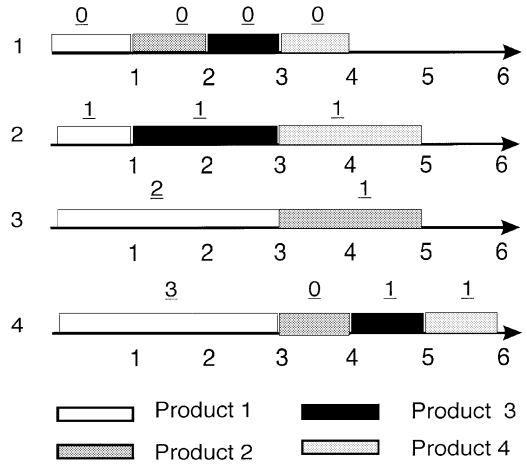

Fig. 1. Result with the elementary approach.

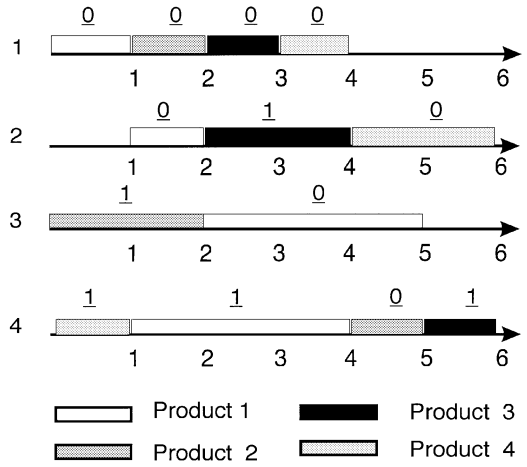

Fig. 2. Result with the event graph approach.

The schedule is represented in Fig. 1. The labels are the underlined integer at the top of the tasks.

The throughput for each product is one item per six time units. Since WIP equals the product of cycle time and throughput (by Little's Law), there are, on average, 21/6 items of product 1 in the system, 10/6 items of product 2, 9/6 items of product 3, and 9/6 items of product 4. The total average WIP is, thus, $(21 + 10 + 9 + 9)/6 = 8.1667$ items.

D. Event Graph Approach

In this approach, the schedule, which is the sequencing of the product types on the machines, is given *a priori*. The algorithm is based on an event graph model of the cyclic manufacturing system. The method is developed in [5] and leads to the result presented in [5, fig. 2]. This result leads to the schedule represented in Fig. 2. For the sake of simplicity, we give the labels at the top of the corresponding tasks. These labels are obtained by applying steps 2–6 of *Algorithm 1*.

We compute

$$Q(1) = 6 + 4 = 10, \quad Q(2) = 6, \quad Q(3) = 6 + 4$$

and

$$Q(4) = 6 + 4 - 3 = 7.$$

Thus, the total average WIP is $(10+6+10+7)/6 = 5.5$ items. The result is better than the one obtained using the elementary approach as far as WIP is concerned.

III. Sufficient Condition for Optimality

Definition: Let $b = 1, \ldots, n$ be the set of products that are manufactured during each cycle, and let $Q(b)$ be the total time necessary to complete one item of product b according to the cyclic schedule under consideration. In other words, $Q(b)$ is the item cycle time. Then, the WIP is defined as $(Q(1) + \cdots + Q(n))/C^*$, where C^* is the duration of one cycle.

Result 1: Let $b = 1, 2, \ldots, n$ be the set of products that are manufactured during each cycle, let C^* be the duration of the cycle, let $Q(b)$ be the item cycle times, and let $H(b)$ be the sum of the durations of the tasks required to produce one item of product b. If, for a given schedule, $Q(b)/C^* \leq \lceil H(b)/C^* \rceil$ for all products b, then the schedule is optimal, in the sense that the maximum throughput is obtained with the minimum WIP. As usual, $\lceil a \rceil$ is the smallest integer greater than or equal to a.

Proof: $\lceil H(b)/C^* \rceil$ is the minimal number of cycle times required to complete a product of type b. Thus, the best schedule possible to complete a product of type b during each cycle is a schedule for which $\lceil Q(b)/C^* \rceil$ products of type b, at different stage of completion, can be observed in the system during each cycle. In other words, the part of the WIP corresponding to b is less than or equal to $\lceil H(b)/C^* \rceil$. Q.E.D.

This result provides the best possible WIP. If a solution is such that *Result 1* holds, then it is optimal. The reverse is not true. This result allows to evaluate the maximal "distance" to the optimal solution.

Consider, for instance, the first product in the example performed using the elementary approach. In this case, $Q(1)/6 = 21/6 = 3.5$ and $\lceil H(1)/6 \rceil = \lceil 8/6 \rceil = 2$. The inequality of *Result 1* does not hold true. Consider the same example solved using the event graph approach. In this case, $Q(1)/6 = 16/6 = 2.667 > \lceil H(1)/6 \rceil = \lceil 8/6 \rceil = 2$, and the inequality of *Result 1* does not hold true either. As a consequence, we cannot claim that one of these cases is optimal.

IV. A New Approach

This section presents an approach that extends the event graph approach presented previously. Called the construction approach, it requires only a sequence of tasks for a bottleneck resource and creates sequences for the other resources.

A. Introduction of the Construction Approach

The construction approach is based on the event graph model of the manufacturing system (see [5]). Since in an event graph each place has only one input transition and one output transition, a place can be denoted either by its name or by its input and output transition. In the rest of this paper, the pair of its input and output transition indexes often denotes a place. For instance, in Fig. 3, p_9 can also be denoted by (9,10) and p_{23} by (13,4).

The event graph approach presented above requires, for each resource, the sequence of tasks to be performed. The construction approach requires a sequence of tasks for the bottleneck resource (or one of the bottleneck resources) only. The algorithm constructs iteratively the sequences for the other resources.

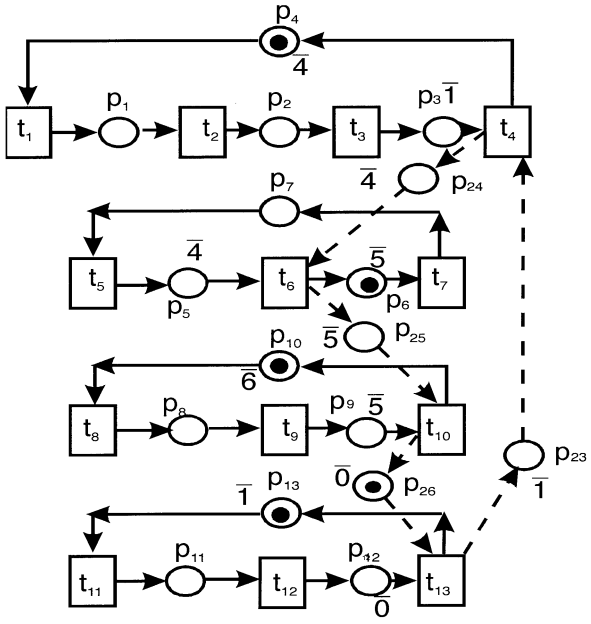

Fig. 3. Process circuit and command circuit of the bottleneck resource.

The idea behind this algorithm is quite simple. Since we want the bottleneck resource to be fully utilized, the token assigned to that resource's command circuit should never wait in a place. As a consequence, as soon as a transition of this command circuit is fired, the next transition should be enabled, and its firing must start immediately. Because each transition has two input places (one belonging to the command circuit and one to a process circuit), the input place belonging to the process circuit should contain at least one token at the time the transition must fire. We introduce this token to this place when it is needed. When a transition firing ends, one token appears in the output place belonging to the process circuit and another token appears in the output place belonging to the command circuit. We continue to fire the transitions in the command circuit under consideration until one cycle is completed. This completes the first stage (stage A) of the algorithm.

At this point of the computation, there is one token in each one of the output places of the transitions of the command circuit under consideration that belong to a process circuit, and the arrival time of the tokens in the output places are known.

The second stage (stage B) of the algorithm simultaneously builds the command circuits of the other resources (i.e., the remainder of the schedule) and determines the start times of the other tasks. We provide some more details concerning this second stage.

If none of the transitions of a process circuit belongs to the command circuit considered in stage A, then one token is added in the input place of the first transition, this input place being the one that belongs to the process circuit [step 4) of the algorithm]. The first transition of the process circuit is the one that corresponds to the first operation of the related manufacturing process. Now, there is one token in each process circuit of the event graph.

From this point on, we fire the transitions that have not been fired yet. We select, in the process circuits, the place (i, j) that contains a token for the longer period of time among all the places that contain a token. $z(i, j)$ is the time the token arrived in this place. The output

transition of this place is denoted by t_j. This transition represents a task k performed by resource $r(k)$. Let F be the set of transitions that represent the tasks performed by $r(k)$.

If no transitions of F have been fired previously, which means that the design of the command circuit related to $r(k)$ did not start yet, we introduce the input place u_j of t_j that will belong to the command circuit related to $r(k)$. We assign one token to u_j at time $w(u_j) = z(i, j)$. Thus, the command circuit related to resource $r(k)$ starts with arc (u_j, t_j).

If some transitions of F have already been fired, the command circuit related to $r(k)$ exists partially. Let t_v be the transition of F that has been fired last. In this case, we introduce a command place u_j that is the output place of t_v and the input place of t_j: two new arcs, (t_v, u_j) and (u_j, t_j), have been added to the command circuit related to $r(k)$.

In either case, it is possible to compute the time the firing of t_j ends, assuming that transitions are fired as soon as possible. This time is stored in $z * (j)$.

Indeed, if t_j is the last transition of F that has been fired and t_m the first, we introduce a command place u_m that is the output place of t_j and the input place of t_m. This completes the command circuit related to $r(k)$ and step 6) of the algorithm.

We then consider transition t_s that follows t_j in the process circuit. Two cases may happen at this point: either a token already visited the place (j, s) previously, or not. In the first case, $z(j, s)$ contains a value which is the last time a token entered place (j, s), otherwise $z(j, s)$ is undefined.

If $z(j, s)$ is undefined, we introduce a token in (j, s), remove one token from the input places, and set $z(j, s) = z * (j)$. Otherwise, if the difference between the time t_j ends firing and the last time a token visited (j, s), that is $z * (j) - z(j, s)$, is less than C^* cycle time of the command circuit related to the bottleneck resource, then we also introduce a token in (j, s) and set $z(j, s) = z * (j)$. If $z * (j) - z(j, s) > C^*$, the solution is not feasible. From a manufacturing point of view, this inequality means that it is impossible to produce one part of the type corresponding to the manufacturing process each C^* period. At least one more unit of this type of part is required in the WIP. We reset the event graph to its initial state, add one token in (i, j), and restart the process, and so on, until it is possible to fire all the transitions without reaching the inequality that shows unfeasibility.

B. Construction Algorithm

This section presents the algorithm. We will give some examples in the next section.

Algorithm 2: Given: A sequence of tasks for a bottleneck resource.

Initialization: Construct all of the process circuits. The values $z(i, j)$ for all places (i, j) in the process circuits are undefined. $z(i, j)$ will contain the last time a token became available in place (i, j). For each transition t_j, construct a second input place u_j. These control places will form the command circuits. The values $w(u_j)$ for all control places are undefined. $w(u_i)$ will contain the last time a token became available in u_i. C^* equals the total processing time on the bottleneck resource.

Part A: Fire the transitions of the command circuit of a bottleneck resource.

1) Construct a command circuit corresponding to the sequence of tasks that bottleneck resource performs. If the resource performs tasks $i_1, i_2, \ldots, i_n$ during each cycle, let $\langle t_{i_1}, t_{i_2}, \ldots, t_{i_n} \rangle$ be the command circuit. Hereafter, let $k + 1 = 1$ if $k = n$, and $k - 1 = n$ if $k = 1$. For $k = 1, \ldots, n$, connect each transition t_{i_k} to the input place u_{i_k+1}. Introduce a token in place u_{i_1} and set $w(u_{i_1}) = 0$ and $z(i_n, i_1) = 0$.
2) For $k = 1, 2, \ldots, n$, do the following steps.
   - 2.1) Let b be the product that requires task i_k. Let t_j be the transition that immediately precedes t_{i_k} in the process circuit for product b. Add a token to place (t_j, t_{i_k}) and set $z(j, i_k) = z(i_{k-1}, i_k)$.
   - 2.2) Fire transition t_{i_k}. Set $z * (i_k) = w(u_{i_k}) + (i_k)$, where (i_k) is the firing time of t_{i_k}. Remove one token from places u_{i_k} and (t_j, t_{i_k}).
   - 2.3) Let t_h be the transition that immediately follows t_{i_k} in the process circuit for product b.
   - 2.4) Add one token to places u_{i_k+1} and (t_{i_k}, t_h). Set $w(u_{i_k+1}) = z * (i_k)$. Set $z(i_k, h) = z * (i_k)$.
3) Let PN be the resulting event graph. Let Z be the set of values $z(.,.)$ applied to places in the process circuits. Let W_g be the set of values $w(.)$ applied to the control places.

Part B. Perform a cycle while building the missing command circuits.

4) If there are any process circuits with no tokens, then, for each such circuit, add one token to the place (i, j) that is the input place for the first transition in that process circuit, and set $z(i, j) = 0$. Mark with the label "NB" all of the places that belong to a process circuit and contain a token, where "NB" stands for "not blocked."
5) Let E be the set of places that belong to a process circuit, contain a token, and are labeled "NB." If E is empty, then go to step 9). Otherwise, select the place (i, j) in E with the minimal $z(i, j)$.
6) Consider transition t_j that represents task v, the resource $r(v)$ that performs task v, and the other tasks that resource $r(v)$ performs. Let F be the set of corresponding transitions.

   If no transitions in F have been fired at this point, t_j is the first transition of the command circuit for $r(v)$. Add a token to u_j. If $w(u_j)$ is undefined, set $w(u_j) = z(i, j)$.

   Otherwise, if t_k is the transition in F that was fired last, t_j follows t_k in the command circuit for $r(v)$. Connect transition t_k to place u_j. Add a token to u_j. Let $w(u_j) = z * (k)$.

   In either case, let $z * (j) = \max\{z(i, j), w(u_j)\} + (j)$.

   If t_j is the only unfired transition in F, then let t_m be the transition (in this command circuit) that was fired first. Connect transition t_j to the place u_m.

   Let t_s be the transition that follows t_j in the process circuit.
7) If $z(j, s)$ is undefined, fire t_j. Remove one token from the place (i, j) and remove one token from the place u_j. Add one token to the place (j, s). Define $z(j, s) = z * (j)$. Remove the label "NB" from (i, j) and apply the label "NB" to (j, s). Go to step 8.3).
8) Otherwise, $z(j, s)$ is defined.
   - 8.1) If $z * (j) \leq z(j, s) + C^*$, then fire t_j. Remove one token from the place (i, j) and remove one token from the place u_j. Add one token to the place (j, s). Redefine $z(j, s) = z * (j)$. Remove the label "NB" from (i, j) and apply the label "NB" to (j, s). This means that the new token in place (j, s) is blocked and cannot enable transition t_s. Go to step 8.3).
   - 8.2) Otherwise, $z * (j) > z(j, s) + C^*$. The solution is infeasible. Reset the event graph to *PN*. Update M by adding a token to place (i, j). Reset the values on the process circuit places to the set Z. Update Z by defining

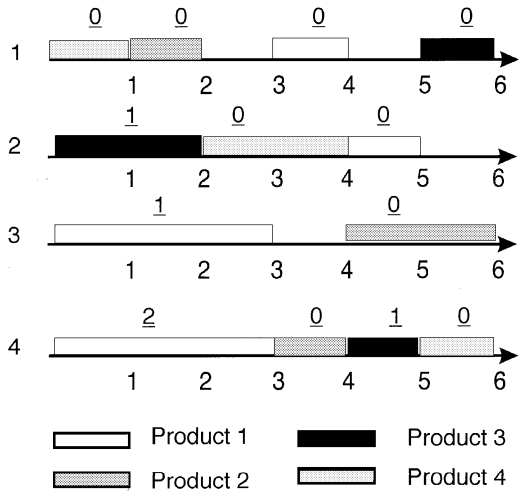

Fig. 4. Schedule using the construction approach.

$z(i, j) = z*(j) - \mu(j) - C^*$. Reset the values on the control places to the set W_g. Return to step 4).

8.3) If t_j was the last unfired transition in F, then let t_m be the transition (in this command circuit) that was fired first. If $z*(j) > w(u_m) + C^*$, then the *PN* was infeasible. Reset the event graph to *PN*. Reset the values on the process circuit places to the set Z. Reset the values on the control places to the set W_g. Update W_g by defining $w(u_m) = z*(j) - C^*$. Return to step 4).

9) Create a feasible schedule as follows. Each task j ends at time $z*(j)$. The task begins at time $z*(j) - (j)$. (Any task that starts outside [0, C^*] can be shifted to this interval by adding or subtracting a multiple of C^*.) Use the elementary approach (*Algorithm 1*) to calculate the cycle time and WIP of the schedule.

In this algorithm, we visit all the transitions to fire them. If the algorithm reaches an unfeasible situation, the whole process restarts after adding one token in the WIP. We know that putting one token in each place of the process circuit is enough to maximize the throughput. Finally, the complexity of *Algorithm 2* is in $0(n_t^2)$, where n_t is the number of transitions in the event graph model.

C. Example

We still consider the example presented in [5] and we keep the same task sequence (4, 1, 2, 3) for the bottleneck resource 4. This task sequence belongs to the solution of the event graph algorithm presented in [5]. In Fig. 3, we present the process circuits and the command circuit corresponding to resource 4. The marking is the one obtained at the end of *Part A* of the algorithm. The upperlined numbers are the arrival times of the tokens in the places, that is the $z(i, j)$ values.

Part B of the algorithm leads first to an unfeasible solution since place (3,4) is marked for the second time at a time greater than $1 + 6 = 7$, where 1 is the first marking time of (3,4) and $C^* = 6$. We then restart the computation of *Part B* of the algorithm after introducing a token in place (2,3) at time 0. This leads to the solution represented in Fig. 4.

In this case, $Q(1) = 2 * 6 = 12$, $Q(2) = 4$, $Q(3) = 6$, and $Q(4) = 6$. The total average WIP is, in this case, $(12+4+6+6)/6 = 28/6 = 4.667$. This result is the best among the ones obtained using the three methods, and $Q(1)/6 = 12/6 = 2 = \lceil H(1)/6 \rceil = 2$, $Q(2)/6 = 4/6 = 0.667 < \lceil H(1)/6 \rceil = 2$, $Q(3)/6 = Q(4)/6 = 1 < \lceil H(1)/6 \rceil = 2$. According to *Result 1*, we can claim that this result is optimal.

V. Conclusion

In this paper, we showed how the problem of maximizing the productivity of a cyclic system evolved. The goal of the elementary approach, which is the first approach that has been proposed, did not take care of the WIP. The WIP was only a parameter used to recreate an order that fit with the manufacturing processes. The event graph method was the first one that tried to maximize the productivity while minimizing the WIP. Unfortunately, the problem was too constrained to reach a good solution. The construction method presented in this paper relaxes most of the constraints. Only the input sequence of the products on a bottleneck resource is required, and the inputs of the products on the other resources are adjusted according to the input sequence of the bottleneck resource. In this construction process, tokens (i.e., WIP units) are introduced only when needed, which explains why most of the results obtained fit with the condition of *Result 1*, and thus, are certainly optimal.

Furthermore, a sufficient condition has been proposed to decide if a schedule is optimal. This condition is based on a realistic hypothesis. If a product is completed during a cycle, it becomes available only at the end of the cycle.

It should be noticed that, to our knowledge, none of the existing heuristics is as efficient as the one proposed in this paper. The algorithm reaches a solution that satisfies the sufficient condition for optimality most of the time. The work made on the example already used in [5] shows the significant improvement provided by the construction method. The test has been made on a Power Macintosh G3 for systems with up to eight machines and ten products, but the size of these examples could be extended.

References

[1] W. Reisig, "Petri nets with individual tokens," *Informatik-Fachberichte*, vol. 66, no. 21, pp. 229–249, 1983.

[2] F. Commoner, A. Holt, S. Even, and A. Pnueli, "Marked directed graphs," *J. Comput. Syst. Sci.*, vol. 5, no. 5, pp. 511–523, 1971.

[3] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, pp. 541–580, Apr. 1989.

[4] P. Chretienne, "Les réseaux de Petri temporisés," Thése d'Etat, Univ. Paris VI, Paris, France, 1983.

[5] S. Laftit, J.-M. Proth, and X.-L. Xie, "Optimization of invariant criteria for event graphs," *IEEE Trans. Automat. Contr.*, vol. 37, pp. 547–555, May 1992.

[6] H. P. Hillion and J.-M. Proth, "Performance evaluation of job-shop systems using timed event graphs," *IEEE Trans. Automat. Contr.*, vol. 34, pp. 3–9, Jan. 1989.

[7] J.-M. Proth and X. L. Xie, *Petri Nets. A Tool for Design and Management of Manufacturing Systems*. New York: Wiley, 1996.

[8] I. Minis and J.-M. Proth, "Production management in a Petri net environment," *Recherche Opérationnelle/Oper. Res.*, vol. 29, no. 3, pp. 321–352, 1995.

[9] A. A. Desrochers and R. Y. Al-Jaar, *Application of Petri Nets in Manufacturing Systems*. Piscataway, NJ: IEEE Press, 1995.

[10] M. C. Zhou and K. Venkatesh, *Modeling, Simulation and Control of Flexible Manufacturing Systems: A Petri Net Approach*. Singapore: World Scientific, 1998.