On-line Scheduling with WIP regulation

Fabrice Chauvet, Jean-Marie Proth, Yorai Wardi

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

On-line Scheduling with WIP regulation

Fabrice Chauvet - Jean-Marie Proth - Yorai Wardi

N° 3432
Mai 1998

THÈME 4

Rapport de recherche

Processus d'ordonnancement en temps réel avec contrôle des en-cours

Fabrice CHAUVET*, Jean-Marie PROTH* and ** and Yorai WARDI ***

RESUME

Le système considéré est composé de plusieurs machines. Certains sous-ensembles de ce système sont composés de machines identiques ou différentes mais capables d'exécuter les mêmes opérations. Le système est automatisé, et il n'est pas possible de stocker des composants ou des produits semi-finis au cours de la fabrication. Une flexibilité limitée du système existe : il est possible d'étendre les temps opératoires dans certaines limites, mais dans ce cas la ressource concernée n'est pas disponible pour exécuter une autre opération. Il n'y a pas de conflit entre les ressources ; en d'autres termes, une même machine n'est jamais utilisée pour exécuter plus d'une opération sur le même produit. Du fait de la fréquence élevée d'arrivée des produits à fabriquer, il n'est pas possible de remettre en question l'ordonnancement des produits déjà ordonnancés. Par conséquent, lorsqu'une nouvelle commande arrive dans le système, il faut tirer avantage au mieux des périodes de vacuité des ressources. L'objectif est d'exécuter la commande au plus tôt. Nous proposons un algorithme qui garantit l'obtention d'un temps de fin de fabrication (makespan) aussi court que possible. Cet algorithme est illustré par des exemples numériques. Il est appliqué en particulier au contrôle des en-cours dans un système d'assemblage.

MOTS CLES

Ordonnancement temps réel, makespan.

* INRIA Lorraine, Technopôle Metz 2000, 4 rue Marconi, 57070 Metz, France
** Institute for Systems Research, University of Maryland, College Park, MD 20742, USA
*** Scholl of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA

On-line Scheduling with WIP regulation

Fabrice CHAUVET*, Jean-Marie PROTH* and ** and Yorai WARDI ***

ABSTRACT

The system under consideration is composed with several machines, and some of these machines may be identical or able to perform the same operations. The manufacturing system is fully automated, and semi-finished products or components are not stored during the process. A limited flexibility exists since the manufacturing times can be extended within certain limits at the expense of the unavailability of the resource in charge of the operation. There are no conflicts between the resources; in other words, the same machine is not used to perform different operations required to complete the same product. Due to the intensity of the flow of products to be manufactured, it is impossible to reschedule the products which have been previously scheduled. Thus, when a new product requirement arrives in the system, we have to take advantage of the idle time windows. The goal is to complete the product as soon as possible. We give a real-time scheduling algorithm which guarantees an optimal makespan to any product which arrives in the system. Some numerical examples are provided to illustrate this approach. In particular, this approach is applied to the regulation of the WIP in an assembly system.

KEYWORDS

Makespan optimization, Real-time scheduling, On-line scheduling.

* INRIA Lorraine, Technopôle Metz 2000, 4 rue Marconi, 57070 Metz, France
** Institute for Systems Research, University of Maryland, College Park, MD 20742, USA
*** Scholl of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA

1. INTRODUCTION

The system considered in this paper is fully automated. Customers' requirements stream in the system, and each new requirement should be scheduled as soon as it enters the system, the goal being to reduce its completion time as much as possible. In other words, the goal is to minimize the makespan of each product considered individually. Due to the importance of the production flow, it is impossible to reschedule a product previously scheduled. In other words, when a customer's requirement arrives in the system, and this can happen at any time, the goal is to take advantage of the idle periods available on the resources to complete the product as soon as possible.

We assume that a component is not allowed to wait in front of a resource: the next operation to be performed on the component should start as soon as the current one ends. This constraint is very strong, mainly in case of assembly, since several components which have to be assembled should be completed exactly at the same time. This constraint aims at reducing the work-in-process (WIP) as much as possible.

Nevertheless, limited flexibility exists: the processing time associated with a given operation may be extented within certain limits, but the resource on which an operation is performed is unavailable for another operation as long as the current operation is not completed. In particular, we can find this kind of limited flexibility in chemical processes.

Numerous papers have been dedicated to no-wait or blocking problems. In these problems, it is assumed that no intermediate buffers exist between the resources.

Callahan [1971] uses a queueing model to study a no-wait processing in the steel industry. Chu and all [1998], and Chauvet and all [1997] consider the surface treament which is a no-wait problem. McCormick and all [1989] study a cyclic flowshop with buffers which can be transformed into a blocking problem by considering buffers as resources with totally flexible processing times, i.e. processing times which can take any value between 0 and $+\infty$. Hall and Sriskandarajah [1996] make a survey on scheduling problems with blocking and no-wait, while Rachamadugu and Stecke [1994] classify the FMS scheduling procedures. This paper should be considered as an intermediate approach: some flexibility exists in the model, but this flexibility can be limited. An application to the regulation of the WIP in an assembly system is proposed. The same regulation can be introduced in any automated production system.

2. NOTATIONS

Each operation of the process is defined by its minimal and maximal processing times. For instance, operation i is characterized by θ_i, minimal processing time, and $\theta_i + \delta_i$, maximal processing time.

Each operation i is characterized by the set of its predecessors denoted by Γ_i^-, and the set of its successors denoted by Γ_i^+. In this paper, $j \in \Gamma_i^-$ means that j ends exactly when i starts. Similarly, $j \in \Gamma_i^+$ means that j ends as soon as i starts. The set of operations which begin at the same time as operation i is denoted by Φ_i^- and the set of operations which end at the same time as i is denoted by Φ_i^+. In other words, if $i \in \Gamma_k^+$, then any $j \in \Gamma_k^+$ belongs to Φ_i^-. Similarly, if $i \in \Gamma_k^-$, then any $j \in \Gamma_k^-$ belongs to Φ_i^+.

Thus, a process **P** is characterized by the list $\{1, 2, \ldots, n\}$ of its operations, each operation $i \in \{1, 2, \ldots, n\}$ being characterized by six elements: $\{\theta_i, \delta_i, \Gamma_i^-, \Phi_i^-, \Phi_i^+, \Gamma_i^+\}$. We denote by b_i the starting time and by e_i the completion time of operation i, $i \in \{1, 2, \ldots, n\}$. At this point, the introduction of Φ_i^- and Φ_i^+ may appear as being somewhat useless since it is possible to use Γ_k^- and Γ_k^+ to define these variables. Nevertheless, these notations will considerably simplify the following explanations.

Note that if $i \in \Gamma_k^-$ and $j \in \Gamma_k^-$, then for any k' such that $i \in \Gamma_{k'}^-$, we have also $j \in \Gamma_{k'}^-$, since no delay is allowed between two consecutive operations. Similarly, if $i \in \Gamma_k^+$ and $j \in \Gamma_k^+$, then for any k' such that $i \in \Gamma_{k'}^+$, we have also $j \in \Gamma_{k'}^+$. Furthermore, since no operation begins at the same time it finishes, $j \notin \Phi_i^-$ if $j \in \Gamma_i^+$, and $j \notin \Phi_i^+$ if $j \in \Gamma_i^-$.

3. PROBLEM FORMULATION

The manufacturing system is composed with several machines, and some of these machines may be identical. Since other products have been scheduled previously, the machines may be partially busy when a new product is introduced in the system. The manufacturing system being fully automated, semi-finished products or components are not stored during the process. The only flexibility available in the system is given by the δ_i values, which express the fact that the manufacturing time θ_i of operation i can be extented to $\theta_i + \delta_i$. This is in some way equivalent to the storage of semi-finished products or components, but makes the machines unavailable for other operations.

For each operation i, we have a series of available windows denoted by: $[\alpha_1^i, \beta_1^i]$, $[\alpha_2^i, \beta_2^i]$, ..., $[\alpha_{q_i}^i, \beta_{q_i}^i]$. These windows may be available on different machines. For each operation i, they are ordered in the increasing order of the α_k^i, $k = 1, 2, \ldots, q_i$. When two α_k^i are equal, the order is the increasing order of the β_k^i. There are no conflicts between the resources; in other words, the same machine cannot be used to perform different operations for a same product. The last window q_i associated with each operation i, is such that $\beta_{q_i}^i = +\infty$.

A feasible solution to the problem associated with a set $[\alpha_{r_i}^i, \beta_{r_i}^i]$ of idle windows (where $i = 1, 2, \ldots, n$ and $r_i = 1, 2, \ldots, q_i$) is a set of operations whose starting and completion times are respectively b_i and e_i, $i = 1, 2, \ldots, n$. These times verify, for $i = 1, 2, \ldots, n$:

$$\begin{cases} \alpha^i_{r_i} \leq b_i & (1) \\ e_i \leq \beta^i_{r_i} & (2) \\ \theta_i \leq e_i - b_i & (3) \\ e_i - b_i \leq \theta_i + \delta_i & (4) \\ b_i = e_j, \ \forall j \in \Gamma_i^- & (5) \\ b_i = b_j, \ \forall j \in \phi_i^- & (6) \\ e_i = e_j, \ \forall j \in \phi_i^+ & (7) \\ e_i = b_j, \ \forall j \in \Gamma_i^+ & (8) \end{cases}$$

Note that inequalities (3) and (4) allow to assign to operation i a processing time $w_i \in [\theta_i, \ \theta_i + \delta_i]$ such that $b_i + w_i = e_i$. A feasible solution is optimal if the makespan $\max\limits_{i/\Gamma_i^+=\varnothing}(e_i)$ is minimal.

The goal is to schedule the operations required to manufacture a product whose process is **P**, in order to minimise the makespan, i.e. to complete the product as soon as possible. This scheduling has to be performed as soon as the product arrives in the system. Due to the intensity of the flow of products, it is not allowed to reschedule products which have been previously scheduled.

NB. The scheduling problems with no-wait and blocking are two particular cases of the problem presented in this paper. In the first case (no-wait problem) $\delta_i=0$ while $\delta_i=+\infty$ in the blocking problem.

4. OPTIMAL SOLUTION OF ACYCLIC PRODUCTION SYSTEM

4.1. MODELING AND TEST OF CYCLICITY

The system at hand can be represented by a precedence graph G where:

(i) Each arc represents one operation, and we call i the arc representing operation i.

(ii) If $j \in \Gamma_i^-$, then the end of the arc representing j is the origin of the arc representing i. This means that operation i should start as soon as operation j ends. Similarly, if $j \in \Phi_i^-$, then the origin of j is the origin of i. In other words, operations i and j start simultaneously. If $j \in \Phi_i^+$, then the end of j is the end of i, i.e. operations i and j end simultaneously. If $j \in \Gamma_i^+$, then the origin of j is the end of i, or j starts as soon as i ends.

The precedence graph G is the set of arcs $\{1, 2, \ldots, n\}$, each arc $i \in \{1, 2, \ldots, n\}$ being connected to the arcs belonging to $\{\Gamma_i^- \cup \Phi_i^- \cup \Phi_i^+ \cup \Gamma_i^+\}$. Thus, G can be defined as follows $G = \left(\{\Gamma_i^-, \ \Phi_i^-, \ \Phi_i^+, \ \Gamma_i^+\}_{i=1,2,\ldots,n}\right)$.

An undirected path of G is a sequence $i_1, i_2, \ldots, i_p$ of p arcs distinct from each other and such that either $i_{k+1} \in (\Gamma_{i_k}^- \cup \Phi_{i_k}^-)$ or $i_{k+1} \in (\Phi_{i_k}^+ \cup \Gamma_{i_k}^+)$ for $k = 1, 2, \ldots, p-1$, and such that i_k and i_{k+2} are not connected to the same end of i_{k+1} for $k = 1, 2, \ldots, p-2$. A cycle of G is an undirected path $i_1, i_2, \ldots, i_p$ (with $p \geq 2$) such that either ($i_1 \in (\Gamma_{i_p}^- \cup \Phi_{i_p}^-)$ and $i_1 \in (\Phi_{i_2}^+ \cup \Gamma_{i_2}^+)$), or ($i_1 \in (\Phi^+_{i_p} \cup \Gamma^+_{i_p})$ and $i_1 \in (\Gamma^-_{i_2} \cup \Phi^-_{i_2})$). A cyclic (resp. acyclic) precedence graph is such that it does (resp. does not) contain a cycle.

The goal of algorithm 1 presented hereafter is to number the arcs of the precedence graph G such that the number associated with an arc i is greater than the numbers associated with the arcs connected with one end of i. Indeed, two arcs will have two different numbers.

Algorithm 1 - Number the arcs of an acyclic precedence graph

```
1. Initialization
        1.1.  $p = 1$
        *p is a counter used to number the arcs.*
        1.2.  For $i = 1, 2, \ldots, n$, set $m(i) = 0$
        *When m(i) = 0, arcs are said to be unmarked. Thus, at the beginning of the algorithm, all the arcs are unmarked.*
2. While there exists i such that $m(i) = 0$ and at least one of the four following conditions is verified:
   (i) $(\Gamma^-_i \cup \Phi^-_i) = \varnothing$
   *The origin of arc i is not connected with another arc.*
   (ii) $(\Phi^+_i \cup \Gamma^+_i) = \varnothing$
   *The end of arc i is not connected with another arc.*
   (iii) for any $j \in (\Gamma^-_i \cup \Phi^-_i)$, $m(j) \neq 0$
   *All the arcs which either end or start when arc i starts are marked.*
   (iv) for any $j \in (\Phi^+_i \cup \Gamma^+_i)$, $m(j) \neq 0$
   *All the arcs which either end or start when arc i ends are marked.*
        2.1.  Set $m(i) = p$
        *Arc i is marked*
        2.2.  Set $p = p+1$
        *The counter is increased by one, which guarantees that the arcs will be marked following the increasing order of the positive integers.*
3. If some of the m(i) are still at 0 then the graph is cyclic, otherwise it is acyclic
```

Remark. The algorithm starts with the initial numbers i assigned to the operations and generate an order m(i) which is required by the approach presented in this paper.

Result 1

If, at the end of algorithm 1, there exists $i \in \{1, 2, \ldots, n\}$ such that $m(i) = 0$, then the graph is cyclic. Otherwise, the graph is acyclic.

Proof:

a. If at the end of algorithm 1, there exists $i \in \{1, 2, \ldots, n\}$ such that $m(i) = 0$, then the graph is cyclic.

Assume that, at the end of algorithm 1, there exists $i \in \{1, 2, \ldots, n\}$ such that $m(i) = 0$. Arc i cannot be marked because there exists $j_1 \in (\Gamma_i^- \cup \Phi_i^-)$ and $k_1 \in (\Phi_i^+ \cup \Gamma_i^+)$ such that $m(j_1) = 0$, $m(k_1) = 0$, $j_1 \neq i$, and $k_1 \neq i$. Let us select j_1. Making the same reasoning, we find $j_2 \in (\Gamma_{j_1}^- \cup \Phi_{j_1}^-)$ such that $m(j_2) = 0$, $j_2 \neq j_1$ and the two arcs i and j_2 are not connected to the same extremety of j_1. In this way, we build an unlimited sequence $i = j_0, j_1, j_2 \ldots$ of arcs. Since the number of arcs in the graph is limited, this sequence will contain two arcs j_r and j_s such that $j_r = j_s$. Thus, the sequence of arcs $j_r, j_{r+1}, \ldots, j_{s-1}$ is a cycle.

b. If at the end of algorithm 1, there doesn't exist $i \in \{1, 2, \ldots, n\}$ such that $m(i) = 0$, then the graph is acyclic.

Assume that the graph G contains a cycle $i_1, i_2, \ldots, i_p$ and that at the end of algorithm 1, there doesn't exist i such that $m(i) = 0$. Since $m(i_p) \neq 0$, we claim that either $m(i_1) < m(i_p)$ or $m(i_{p-1}) < m(i_p)$ to be allowed to mark i_p. If $m(i_1) < m(i_p)$, it turns out that $m(i_2) < m(i_1)$ and, step by step, we find that $m(i_p) < m(i_{p-1}) < \ldots < m(i_2) < m(i_1) < m(i_p)$. Similarly, if $m(i_{p-1}) < m(i_p)$, we obtain $m(i_p) < m(i_1) < \ldots < m(i_{p-2}) < m(i_{p-1}) < m(i_p)$. In both cases, we obtain $m(i_p) < m(i_p)$, which is impossible.

This completes the proof.

QED

Result 1 etablishes that algorithm 1 converges, even if the precedence graph is not acyclic. If the graph is acyclic, each arc i verifies at least one of the two following conditions:

(i) Either each operation j which begins or finishes when the operation i **begins**, is such that $m(j) < m(i)$.

(ii) Or each operation j which begins or finishes when the operation i **ends**, is such that $m(j) < m(i)$.

Result 2

The complexity of algorithm 1 is in o(n).

Proof:

A node of the graph is one of the ends of an arc. The common end of two arcs connected to each other is considered as a unique node. A way to work out algorithm 1 is to weight each node of the graph with the number of arcs which end or start at this node. An arc $i \in \{1, 2, \ldots, n\}$ can be marked if one of its ends is weighted at 1 since, in this case, all the arcs which end or start at this end are already marked, except the arc under consideration. When marking an arc i, we substract one to the weights of each of its ends. We continue the process, selecting at each step one arc having one of its ends weighted at 1. At most n steps are necessary to complete the process. Note that, each time we substract one to the weight of each of the ends of an arc, we test if one of these weight is equal to one. If yes, we store this arc in a particular list which contains the arcs ready for marking. Thus, the selection of an arc which can be marked is straight foreward. Finally, algorithm 1 consists in:

- substracting 1 to two weights at most n times,
- storing an arc in a special list at most n times.

Thus, result 2 holds.

QED

4.2. COMPUTING AN OPTIMAL SOLUTION

In the remainder of this section, we assume that the precedence graph is acyclic and we use algorithm 1 to number the arcs. The numbers assigned to the operations are given in figure 1. Then, we apply algorithm 2, which is the matter of this subsection, in order to obtain an optimal solution, i.e. a solution which minimize the makespan.

The idea behind the algorithm 2 is quite simple and can be explained using figure 1. Let us assume that the window to which each operation is assigned is known.

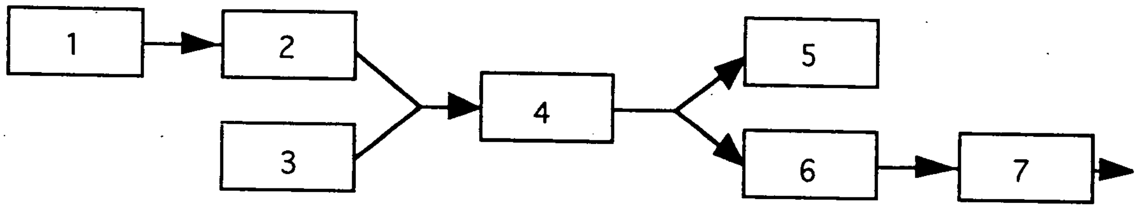


Figure 1. A process composed with seven operations

In this process, there is an assembly-desassembly operation which is operation 4. The other ones are processing operations. Assume that the window where operation i should be located is $\left[\alpha^i_{r_i}, \beta^i_{r_i}\right]$, for $i \in \{1, 2, ..., 7\}$.

We compute a_i, lower bound of the starting time of operation i, and d_i, lower bound of the completion time of operation i, for $i \in \{1, 2, ..., 7\}$. We obtain successively:

- **operation 1**: $a_1 = \alpha^1_{r_1}$ and, since θ_1 is its minimal processing time, $d_1 = a_1 + \theta_1$,
- **operation 2**: Since operation 1 must be finished before the beginning of operation 2, $a_2 = \max\left(\alpha^2_{r_2}, d_1\right)$ and $d_2 = a_2 + \theta_2$,
- **operation 3**: $a_3 = \alpha^3_{r_3}$ and $d_3 = a_3 + \theta_3$,
- **operation 4**: $a_4 = \max\left(\alpha^4_{r_4}, d_2, d_3\right)$ and $d_4 = a_3 + \theta_3$.

For operations 1 to 4, we computed a_i, the lower bound of the starting times of the operations first and, based on these bounds, we computed d_i, the lower bound of the completion time of the operations. The problem is quite different when considering operation 5 since a_5, the lower bound of the starting time of this operation does not depend only on d_4, but also on a_6, the lower bound of the starting time of operation 6, which is unknown at this stage of the computation. It is the reason why we reverse the way of the computation process and compute d_5, the lower bound of the completion time of operation 5, first. We obtain:

- **operation 5**: $d_5 = \alpha^5_{r_5} + \theta_5$ and, since $\theta_5 + \delta_5$ is its maximal processing time, $a_5 = \max(\alpha^5_{r_5},\ d_5 - \theta_5 - \delta_5)$,
- **operation 6**: Since operations 5 and 6 must start at the same time, $a_6 = \max(\alpha^6_{r_6},\ d_4,\ a_5)$ and $d_6 = a_6 + \theta_6$,
- **operation 7**: $a_7 = \max(\alpha^7_{r_7},\ d_6)$ and $d_7 = a_7 + \theta_7$.

We then apply a backward process starting from d_7 to define a set of feasible starting times of the operations. This process leads successively to b_i, minimal starting time of operation i, and to e_i, minimal completion time of operation i, for $i \in \{1, 2, \dots, 7\}$. Indeed, $a_i \le b_i$ and $d_i \le e_i$. Applying the backward process, we obtain successively:

- **operation 7**: $e_7 = d_7$ and $b_7 = a_7$,
- **operation 6**: Since operation 6 must end when operation 7 starts, $e_6 = \max(d_6,\ b_7)$ and, since $\theta_6 + \delta_6$ is its maximal processing time, $b_6 = \max(a_6,\ e_6 - \theta_6 - \delta_6)$,
- **operation 5**: $b_5 = \max(a_5,\ b_6)$ and, since θ_5 is its minimal processing time, $e_5 = \max(d_5,\ b_5 + \theta_5)$,
- **operation 4**: $e_4 = \max(d_4,\ b_6)$ and $b_4 = \max(a_4,\ e_4 - \theta_4 - \delta_4)$,
- **operation 3**: $e_3 = \max(d_3,\ b_4)$ and $b_3 = \max(a_3,\ e_3 - \theta_3 - \delta_3)$,
- **operation 2**: $e_2 = \max(d_2,\ b_4)$ and $b_2 = \max(a_2,\ e_2 - \theta_2 - \delta_2)$,
- **operation 1**: $e_1 = \max(d_1,\ b_2)$ and $b_1 = \max(a_1,\ e_1 - \theta_1 - \delta_1)$.

If operation i cannot be completed before $\beta^i_{r_i}$, i.e. if $e_i > \beta^i_{r_i}$, we replace window $[\alpha^i_{r_i},\ \beta^i_{r_i}]$ by the next one, that is $[\alpha^i_{r_i+1},\ \beta^i_{r_i+1}]$, and we restart the process. We can speed up the computation replacing the window $[\alpha^i_{r_i},\ \beta^i_{r_i}]$ by the first window $[\alpha^i_{r_i+k_i},\ \beta^i_{r_i+k_i}]$ (i.e. k_i is as low as possible and $k_i \ge 1$) such that the operation i can be completed before $\beta^i_{r_i+k_i}$. If all the operations can be completed in the windows to which they are assigned, then the b_i' values are the solution of the problem. This simple example illustrate the general algorithm proposed hereafter.

Algorithm 2 - Acyclic precedence

In the remainder of this section, the operations are labelled by the order m(i) obtain by applying algorithm 1. In other words, i should be understood as m(i).

1. Initialization

    1.1. For $i \in \{1, 2, \dots, n\}$, set $r_i = 1$

    r_i is the rank of the idle time window which is under consideration to perform operation i.

    1.2. For $i \in \{1, 2, \dots, n\}$, set $f(i) = 0$

    f(i) is the operation on which the computation of the schedule of operation i will be based.

2. Foreward process

Computation of lower bounds for the starting times and the completion times of operations.

2.1. For $i = 1$ to n do,

2.1.1. Set $E_i = \{i+1,\ i+2,\ \ldots,\ n\}$

E_i is the set of arcs which have not been consider yet.

2.1.2. $a_i = \alpha^i_{r_i}$

a_i is a lower bound of the starting time of operation i. Initially, a_i is set at the value of the lower limit of the window corresponding to operation i.

2.1.3. $d_i = \alpha^i_{r_i} + \theta_i$

d_i is a lower bound of the completion time of operation i. Initially, d_i is set at the value of the lower limit of the window corresponding to operation i increased by the lower value of the manufacturing time.

2.1.4. If $(\Gamma_i^- \cup \Phi_i^-) \cap E_i = \varnothing$ then

This condition means that, for each operation j which begins or finishes when the operation i begins, a_j and d_j have been previously computed.

2.1.4.1. $a_i = \max\left(a_i,\ \max\limits_{j \in \Gamma_i^-}(d_j)\ \max\limits_{j \in \Phi_i^-}(a_j)\right)$

The starting time a_i is the maximal value among, (i) a lower bound of a_i, (ii) the greater completion time of the operations which preceed i, and (iii) the greater starting time of the operations which are supposed to start at the same time as i.

2.1.4.2. $d_i = \max(d_i,\ a_i + \theta_i)$

The completion time d_i is the maximal value among, (i) a lower bound of d_i, and (ii) the sum of the starting time of operation i and the minimal processing time of i.

2.1.4.3. For $j \in (\Gamma_i^- \cup \Phi_i^-)$ do $f(j) = i$

Any operation j which begins or finishes when the operation i begins, will be scheduled taking into account the starting time of operation i.

2.1.5. If $(\Phi_i^+ \cup \Gamma_i^+) \cap E_i = \varnothing$ then

This condition means that, for each operation which begins or finishes when the operation i finishes, a_j and d_j have been previously computed.

2.1.5.1. $d_i = \max\left(d_i,\ \max\limits_{j \in \Phi_i^+}(d_j)\ \max\limits_{j \in \Gamma_i^+}(a_j)\right)$

The completion time d_i is the maximal value among, (i) a lower bound of d_i, (ii) the greater completion time of the operations which are supposed to end at the same time as i, and (iii) the greater starting time of the operations which succeed to i.

2.1.5.2. $a_i = \max(a_i,\ d_i - \theta_i - \delta_i)$

The starting time a_i is the maximal value among, (i) a lower bound of a_i, and (ii) the completion time of operation i decreased by the maximal processing time of i.

2.1.5.3. For $j \in (\Phi_i^+ \cup \Gamma_i^+)$ do $f(j) = i$

Any operation j which begins or finishes when the operation i finishes, will be scheduled taking into account the completion time of operation i.

3. Backward process

Computation of b_i, the minimal starting time, and e_i, the minimal completion time of operation $i \in \{1, 2, ..., n\}$.

```
3.1. For i = n down to 1 do,
    3.1.1. If $f(i) = 0$ then
    Operation i is scheduled before any operation which begins or finishes when operation i begins or finishes.
        3.1.1.1. $b_i = a_i$
        3.1.1.2. $e_i = d_i$
    3.1.2. If $f(i) \in \Gamma_i^-$ then
    Operation i is scheduled taking into account to $e_{f(i)}$, minimal completion time of operation f(i) which has been previously scheduled and which ends when operation i begins.
        3.1.2.1. $b_i = \max(a_i, e_{f(i)})$
        3.1.2.2. $e_i = \max(d_i, b_i + \theta_i)$
    3.1.3. If $f(i) \in \Phi_i^-$ then
    Operation i is scheduled taking into account $b_{f(i)}$, minimal starting time of operation f(i) which has been previously scheduled and which begins when operation i begins.
        3.1.3.1. $b_i = \max(a_i, b_{f(i)})$
        3.1.3.2. $e_i = \max(d_i, b_i + \theta_i)$
    3.1.4. If $f(i) \in \Phi_i^+$ then
    Operation i is scheduled taking into account $e_{f(i)}$, minimal completion time of operation f(i) which has been previously scheduled and which ends when operation i ends.
        3.1.4.1. $e_i = \max(d_i, e_{f(i)})$
        3.1.4.2. $b_i = \max(a_i, e_i - \theta_i - \delta_i)$
    3.1.5. If $f(i) \in \Gamma_i^+$ then
    Operation i is scheduled taking into account $b_{f(i)}$, minimal starting time of operation f(i) which has been previously scheduled and which begins when operation i ends.
        3.1.5.1. $e_i = \max(d_i, b_{f(i)})$
        3.1.5.2. $b_i = \max(a_i, e_i - \theta_i - \delta_i)$
```

```
4. Test
    4.1. For $i = 1$ to $n$ do,
        4.1.1. While $e_i > \beta^i_{r_i}$, set $r_i = r_i + 1$
        Each operation i must be completed before $\beta^i_{r_i}$. It is why we have to try the next time window.
    4.2. If none of the $r_i$ has been modified, then
        4.2.1. $\max_{i/\Gamma_i^+=\emptyset}(e_i)$ is the minimal makespan
        4.2.2. $b_i$ are the earliest starting time of the operations
    4.3. Else
        4.3.1 Return to step 2.
```

In algorithm 2, it is assumed taht, for each operation i, the corresponding windows are ordered in the increasing order of $\alpha^i_{r_i}$ and, in case of equality, in the increasing order of $\beta^i_{r_i}$. Under this assumption, we see that:

- In the forward process, a_i and d_i are computed once for each operation i, and used once to compute $a_{f(i)}$ and $d_{f(i)}$; so, the complexity is in o(n).
- In the backward process, we compute a_i and d_i once once for each operation i, which leads to a complexity is in o(n), too.
- The previous computation is performed at most q times, where $q = \sum_{i=1}^{n} q_i$ is the total number of idle windows.

Finally, the complexity of the whole algorithm is in o(qn).

Results 2 and 3 presented hereafter guarantee that the solution obtained by applying this algorithm is optimal.

Result 2

Algorithm 2 leads to a feasible solution.

Proof:

a. We start by proving the results (9)-(14) which will be used in the second part of the proof.

Considering steps 3.1.1.1, 3.1.2.1, 3.1.3.1, 3.1.4.2 and 3.1.5.2 in algorithm 2, we derive: $a_i \le b_i$, $\forall i \in \{1, 2, \ldots, n\}$. (9)

Considering steps 3.1.1.2, 3.1.2.2, 3.1.3.2, 3.1.4.1 and 3.1.5.1 in algorithm 2, we derive: $d_i \le e_i$, $\forall i \in \{1, 2, \ldots, n\}$. (10)

Assume that $f(i) \in \Gamma_i^-$, then $i \in \Gamma^+_{f(i)}$. Thus, according to step 2.1.5.1 in algorithm 2, $a_i \le d_{f(i)}$. Since $d_{f(i)} \le e_{f(i)}$ (see relations (10)), we can write $a_i \le e_{f(i)}$. Considering step

3.1.2.1 in algorithm 2, it leads to:

$b_i = e_{f(i)}$, $\forall i \in \{1, 2, \ldots, n\}$ such that $f(i) \in \Gamma_i^-$. (11)

Assume that $f(i) \in \Phi_i^-$, then $i \in \Phi_{f(i)}^-$. Thus, according to step 2.1.4.1 in algorithm 2, $a_i \le a_{f(i)}$. Since $a_{f(i)} \le b_{f(i)}$ (see relations (9)), we can write $a_i \le b_{f(i)}$. Considering step 3.1.3.1 in algorithm 2, it leads to:

$b_i = b_{f(i)}$, $\forall i \in \{1, 2, \ldots, n\}$ such that $f(i) \in \Phi_i^-$. (12)

Assume that $f(i) \in \Phi_i^+$, then $i \in \Phi_{f(i)}^+$. Thus, according to step 2.1.5.1 in algorithm 2, $d_i \le d_{f(i)}$. Since $d_{f(i)} \le e_{f(i)}$ (see relations (10)), we can write $d_i \le e_{f(i)}$. Considering step 3.1.4.1 in algorithm 2, it leads to:

$e_i = e_{f(i)}$, $\forall i \in \{1, 2, \ldots, n\}$ such that $f(i) \in \Phi_i^+$. (13)

Assume that $f(i) \in \Gamma_i^+$, then $i \in \Gamma_{f(i)}^-$. Thus, according to step 2.1.4.1 in algorithm 2, $d_i \le a_{f(i)}$. Since $a_{f(i)} \le b_{f(i)}$ (see relations (9)), we can write $d_i \le b_{f(i)}$. Considering step 3.1.5.1 in algorithm 2, it leads to:

$e_i = b_{f(i)}$, $\forall i \in \{1, 2, \ldots, n\}$ such that $f(i) \in \Gamma_i^+$. (14)

b. We now etablish conditions (1) to (8) which define the feasibility.

b.1. We prove that $\alpha_{r_i}^i \le b_i$, $\forall i \in \{1, 2, \ldots, n\}$. (inequalities (1))

According to steps 2.1.2, 2.1.4.1 and 2.1.5.2 in algorithm 2, we have: $\alpha_{r_i}^i \le a_i$, $\forall i \in \{1, 2, \ldots, n\}$. Furthermore, according to relations (9), $\alpha_{r_i}^i \le b_i$: this leads to inequalities (1).

b.2. We prove that $e_j \le \beta_{r_i}^i$, $\forall i \in \{1, 2, \ldots, n\}$. (inequalities (2))

Since we set $r_i = r_i + 1$ when $e_i > \beta_{r_i}^i$ (see step 4.1.1 in algorithm 2), and since the last window q_i associated with each operation i is such that $\beta_{q_i}^i = +\infty$, then the first solution obtained verifies: $e_i \le \beta_{r_i}^i$: this proves inequalities (2).

b.3. We prove that $\theta_i \le e_j - b_i$, $\forall i \in \{1, 2, \ldots, n\}$. (inequalities (3))

Case 1: If $f(i) = 0$, steps 3.1.1.1 and 3.1.1.2 in algorithm 2 lead to $b_i = a_i$ and $e_i = d_i$. Furthermore, if $f(i) = 0$, then $(\Gamma_i^- \cup \Phi_i^- \cup \Phi_i^+ \cup \Gamma_i^+) \cap \{i+1, i+2, \ldots, n\} = \varnothing$ and $a_i + \theta_i \le d_i$ (see steps 3.2.4.2 and 3.2.5.1). As a consequence, $b_i + \theta_i \le e_i$.

Case 2: If $f(i) \in \Gamma_i^-$ or $f(i) \in \Phi_i^-$, then according to steps 3.1.2.2 and 3.1.3.2: $e_i \ge b_i + \theta_i$.

Case 3: If $f(i) \in \Phi_i^+$ or $f(i) \in \Gamma_i^+$, then we derive from steps 3.1.4.2 and 3.1.5.2: $b_i + \theta_i = \max(a_i + \theta_i,\ e_i - \delta_i)$. Thus, $b_i + \theta_i \le \max(a_i + \theta_i,\ e_i)$. Furthermore, if $f(i) \in \Phi_i^+$ or $f(i) \in \Gamma_i^+$, then $(\Phi_i^+ \cup \Gamma_i^+) \cap \{i+1, i+2, \ldots, n\} \ne \varnothing$, $(\Phi_i^- \cup \Gamma_i^-) \cap \{i+1, i+2, \ldots, n\} = \varnothing$ and $a_i + \theta_i \le d_i$ (see step 2.1.4.2). As a consequence, $b_i + \theta_i \le \max(d_i,\ e_i)$. Using relations (10), we obtain: $b_i + \theta_i \le e_i$.

This proves inequalities (3).

b.4. We prove that $e_j - b_i \le \theta_i - \delta_i$, $\forall i \in \{1, 2, \ldots, n\}$. (inequalities (4))

Case 1: If $f(i)=0$, steps 3.1.1.1 and 3.1.1.2 in algorithm 2 lead to $b_i = a_i$ and $e_i = d_i$. Furthermore, if $f(i)=0$, then $(\Gamma_i^- \cup \Phi_i^- \cup \Phi_i^+ \cup \Gamma_i^+) \cap \{i+1, i+2, \ldots, n\} = \varnothing$ and $d_i - \theta_i - \delta_i \le a_i$ (see steps 3.2.5.2). As a consequence, $e_i - \theta_i - \delta_i \le b_i$.

Case 2: If $f(i) \in \Gamma_i^-$ or $f(i) \in \Phi_i^-$, then we derive from steps 3.1.2.2 and 3.1.3.2: $e_i - \theta_i - \delta_i = \max(d_i - \theta_i - \delta_i,\ b_i - \delta_i)$. Thus, $e_i - \theta_i - \delta_i \le \max(d_i - \theta_i - \delta_i,\ b_i)$. Furthermore, if $f(i) \in \Gamma_i^-$ or $f(i) \in \Phi_i^-$, then $(\Gamma_i^- \cup \Phi_i^-) \cap \{i+1, i+2, \ldots, n\} \ne \varnothing$, $(\Phi_i^+ \cup \Gamma_i^+) \cap \{i+1, i+2, \ldots, n\} = \varnothing$ and $d_i - \theta_i - \delta_i \le a_i$ (see step 2.1.5.2). As a consequence, $e_i - \theta_i - \delta_i \le \max(a_i,\ b_i)$. Using relations (9), we obtain: $e_i - \theta_i - \delta_i \le b_i$.

Case 3: If $f(i) \in \Phi_i^+$ or $f(i) \in \Gamma_i^+$, then according to steps 3.1.4.2 and 3.1.5.2: $b_i \ge e_i - \theta_i - \delta_i$.

This proves inequalities (4).

b.5. We prove that $b_i = e_j,\ \forall j \in \Gamma_i^-,\ \forall i \in \{1, 2, \ldots, n\}$. **(inequalities (5))**

For any $j \in \Gamma_i^-$, if $f(j) = i$, according to relations (14), $e_j = b_{f(j)} = b_i$.

For any $j \in \Gamma_i^-$, if $f(i) = j$, according to relations (11), $e_j = e_{f(i)} = b_i$.

For any $j \in \Gamma_i^-$ such that $i \in \Phi_{f(i)}^-$ and $f(i) \ne i$, $j \in \Gamma_{f(i)}^-$. Thus, considering step 2.1.4.3, $f(j) = f(i)$. Using relations (12) and (14), we obtain: $e_j = e_{f(j)} = b_{f(i)} = b_i$.

For any $j \in \Gamma_i^-$ such that $i \in \Gamma_{f(i)}^+$ and $f(i) \ne j$, $j \in \Phi_{f(i)}^+$. Thus, considering step 2.1.5.3, $f(j) = f(i)$. Using relations (11) and (13), we obtain: $e_j = b_{f(j)} = b_{f(i)} = b_i$.

This proves inequalities (5).

b.6. We prove that $b_i = b_j,\ \forall j \in \Phi_i^-,\ \forall i \in \{1, 2, \ldots, n\}$. **inequalities (6))**

For any $j \in \Phi_i^-$, if $f(j) = i$ or $f(i) = j$, according to relations (12), $b_j = b_i$.

For any $j \in \Phi_i^-$ such that $i \in \Phi_{f(i)}^-$, $f(i) \ne i$ and $f(i) \ne j$, $j \in \Phi_{f(i)}^-$. Thus, considering step 2.1.4.3, $f(j) = f(i)$. Using relations (11), we obtain: $b_j = e_{f(j)} = e_{f(i)} = b_i$.

For any $j \in \Phi_i^-$ such that $i \in \Gamma_{f(i)}^+$, $j \in \Gamma_{f(i)}^+$. Thus, considering step 2.1.5.3, $f(j) = f(i)$. Using relations (12), we obtain: $b_j = b_{f(j)} = b_{f(i)} = b_i$.

This proves inequalities (6).

b.7. We prove that $e_i = e_j,\ \forall j \in \Phi_i^+,\ \forall i \in \{1, 2, \ldots, n\}$. **(inequalities (7))**

For any $j \in \Phi_i^+$, if $f(j) = i$ or $f(i) = j$, according to relations (13), $e_j = e_i$.

For any $j \in \Phi_i^+$ such that $i \in \Gamma_{f(i)}^-$, $j \in \Gamma_{f(i)}^-$. Thus, considering step 2.1.4.3, $f(j) = f(i)$. Using relations (14), we obtain: $e_j = b_{f(j)} = b_{f(i)} = e_i$.

For any $j \in \Phi_i^+$ such that $i \in \Phi_{f(i)}^+$, $f(i) \ne i$ and $f(i) \ne j$, $j \in \Phi_{f(i)}^+$. Thus, considering step 2.1.5.3, $f(j) = f(i)$. Using relations (13), we obtain: $e_j = e_{f(j)} = e_{f(i)} = e_i$.

This proves inequalities (7).

b.8. We prove that $e_i = b_j,\ \forall j \in \Gamma_i^+,\ \forall i \in \{1, 2, \ldots, n\}$. **(inequalities (8))**

For any $j \in \Gamma_i^+$, if $f(j) = i$, according to relations (11), $b_j = e_{f(j)} = e_i$.

For any $j \in \Gamma_i^+$, if $f(i) = j$, according to relations (14), $b_j = b_{f(i)} = e_i$.

For any $j \in \Gamma_i^+$ such that $i \in \Gamma_{f(i)}^-$ and $f(i) \ne j$, $j \in \Phi_{f(i)}^-$. Thus, considering step 2.1.4.3, $f(j) = f(i)$. Using relations (12) and (14), we obtain: $b_j = e_i$.

For any $j \in \Gamma_i^+$ such that $i \in \Phi_{f(i)}^+$ and $f(i) \neq i$, $j \in \Gamma_{f(i)}^+$. Thus, considering step 2.1.5.3, $f(j) = f(i)$. Using relations (11) and (13), we obtain: $b_j = e_{f(j)} = e_{f(i)} = e_i$.

This proves inequalities (8).

QED

Result 3

In the first feasible solution obtained by applying algorithm 2, all the completion times, and the starting times of the operations, are minimal. As a consequence, this solution is optimal.

Proof:

a. For a given set of idle windows we prove that, if there exits a feasible solution, the solution obtained by applying steps 2 and 3 in algorithm 2 is such that no other solution leads to a set of completion times which are lower.

Considering steps 2.1.2 to 2.1.5 in algorithm 2, and keeping in mind that θ_i and $\theta_i + \delta_i$ are respectively the minimal and the maximal processing time for operation i, we see that the a_i values and the d_i values cannot be reduced. As a consequence of steps 3.1.1 to 3.1.5, we claim that the b_i values and e_i values given by algorithm 2 cannot be reduced either.

As a first consequence, if the solution obtained by applying steps 2 and 3 of algorithm 2 to a given set of windows $\left\{\left[\alpha_{r_i}^i, \beta_{r_i}^i\right]\right\}_{i=1, 2, \dots, n}$ does not verify constraint (2) for at least one $i \in \{1, 2, \dots, n\}$, no feasible solution exists for this set of windows.

As a second consequence, if the solution obtained is feasible, it is composed with the lower possible b_i and e_i, starting and completion times of the operations i, for this set of idle windows. Thus, the makespan cannot be reduced, for this set of idle windows.

This completes the first part of the proof.

b. We prove that the first feasible solution obtained by applying algorithm 2 is optimal.

If the solution derived from the set of windows having the minimal lower bounds $\alpha_{r_i}^i$ is feasible, it is optimal, and the starting time of the operations are minimal for this set of windows (see a.). Furthermore, since the completion times increase with the $\alpha_{r_i}^i$ values, the solution is optimal and the starting time are minimal for any set of idle windows $\left\{\left[\alpha_{r_i}^i, \beta_{r_i}^i\right]\right\}_{i=1, 2, \dots, n}$.

Moreover, if the operation i is such that $e_i > \beta_{r_i}^i$, then any set of windows containing $\left[\alpha_{r_i}^i, \beta_{r_i}^i\right]$ will lead to the same inequality and thus will not lead to a feasible solution.

This completes the proof.

QED

5. EXAMPLES

Two examples are given in this section. The first example is introduced to illustrate algorithm 2. The second example show how to use the approach presented in this paper for Work-In-Process (WIP) regulation in automated system.

5.1. ILLUSTRATE EXAMPLE

The process represented in Figure 1 is composed with 7 operations. Each operation is defined by θ_i, its minimal processing time, and $\theta_i + \delta_i$, its maximal processing time. Since the system is fully automated, components cannot be stored in front of the resources. In this example, operations (except operation 1, which is a chemical treatment) can be extented as much as necessary, but in this case, machines are unavailable for other operations until the current operation stops.

Table 1 - Processing times

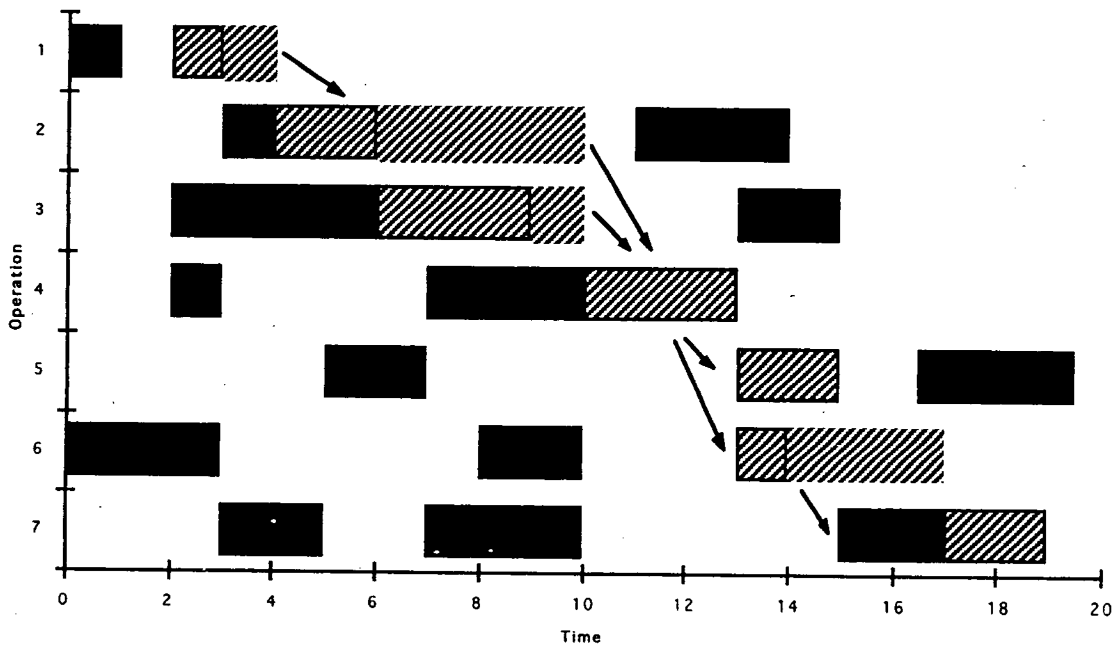
i	1	2	3	4	5	6	n = 7
θ_i	1.	2.	3.	3.	2.	1.	2.
δ_i	1.	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$

Each operation is assigned to a given machine. Since severals operations are previously scheduled, these 7 operations must be performed in one of the idle periods $\left[\alpha^i_{r_i}, \beta^i_{r_i}\right]$ available on the resource.

Table 2 - Available windows

i	1	2	3	4	5	6	n = 7
q_i	1	3	3	3	3	2	4
$[\alpha^i_1, \beta^i_1]$	[1, +∞[	[0, 3]	[0, 2]	[0, 2]	[0, 5]	[3, 8]	[0, 3]
$[\alpha^i_2, \beta^i_2]$		[4, 11[	[6, 13]	[3, 7]	[7, 17]	[10, +∞[	[5, 7]
$[\alpha^i_3, \beta^i_3]$		[14, +∞[	[15, +∞[	[10, +∞[	[20, +∞[		[10, 15]
							[17, +∞[

By applying algorithm 2, we obtain the optimal solution in which the starting times of operations are minimal.



■ Busy periods (due to products previously scheduled)

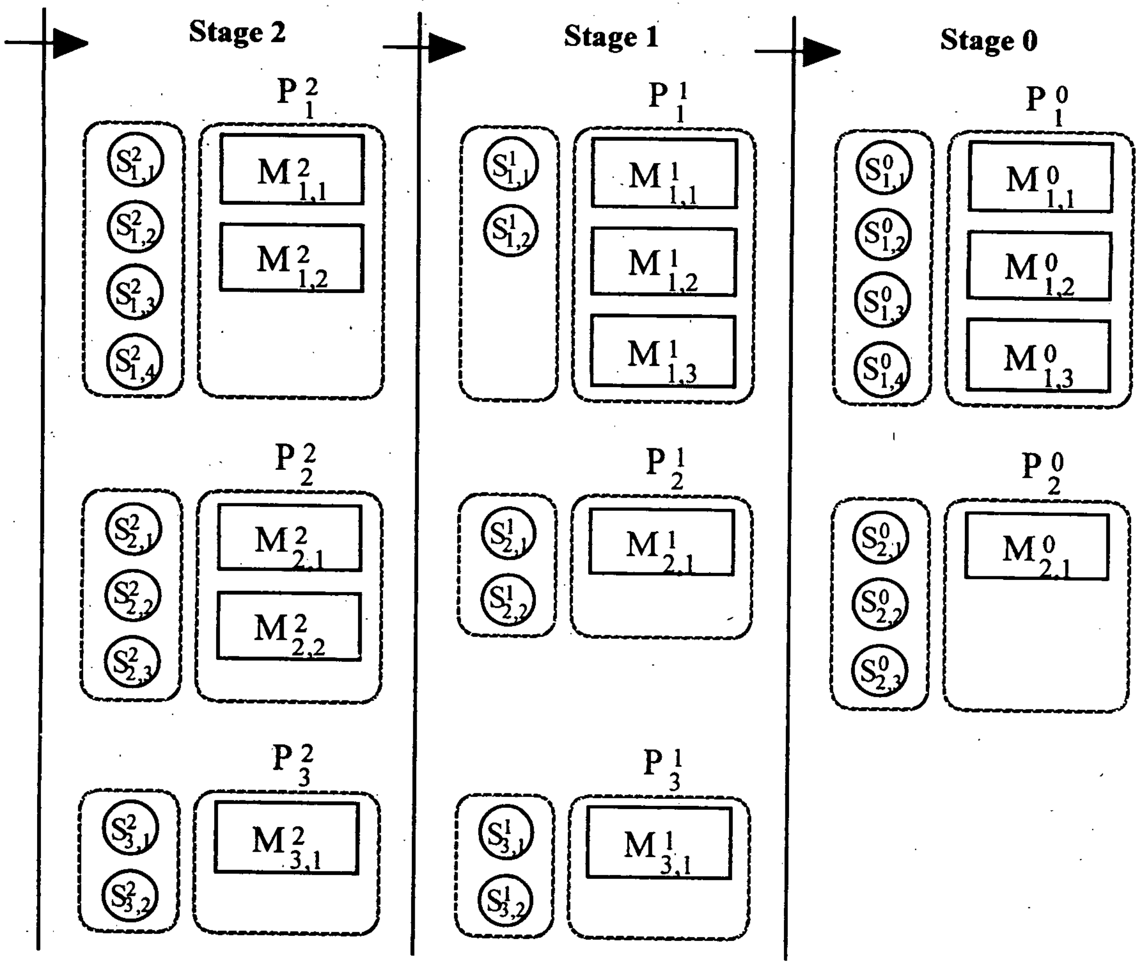
▨ Operations scheduled by applying algorithm 2 (minimal processing time)

▨ Operations scheduled by applying algorithm 2 (selected processing time)

Figure 2. Optimal schedule of the process

5.2. WIP REGULATION

In this section, we consider an assembly system which includes K+1 production stages. Note that the approach could be applied to a more general type of automated production system. Each production stage k (k = 0, 1, ..., K) is composed with p^k pools of machines, each pool $p = 1, 2, \ldots, p^k$ containing m_p^k machines. s_p^k storage entities are ready to store one unit of product in front of the p-th pool of the k-th production stage. If the pool is composed with assembly resources, each storage entity can store all the components required to assemble one unit of the product or semi-finished product resulting from the assembly operation. Figure 3 represents such a system. The production stages are numbered backward from 0 (final production stage) to K. If several components concerning the same product are manufactured at the same production stage, they are manufactured by different pools of machines.



P^k_p is the p-th pool at stage k

$M^k_{p,m}$ is the m-th machine of the p-th pool at stage k

$S^k_{p,s}$ is the s-th storage entity available in front of the p-th pool at stage k

Figure 3. An automated assembly process

The assembly process represented in Figure 4 can be manufactured on the assembly system represented in Figure 3. The pools P^k_p associated with the operations are the pools of machines required to perform the operations.

In such an assembly system, the processing time associated with each operation is fixed (i.e. $\delta_i = 0$ for any manufacturing or assembly operation i). Each storage entity is considered as a resource. The minimal processing time for such a resource is equal to zero (i.e. $\theta_i = 0$ if i is a "storage operation") and the maximal processing time is the same for each storage operation (i.e. $\delta_i = \delta$ for any storage operation i).

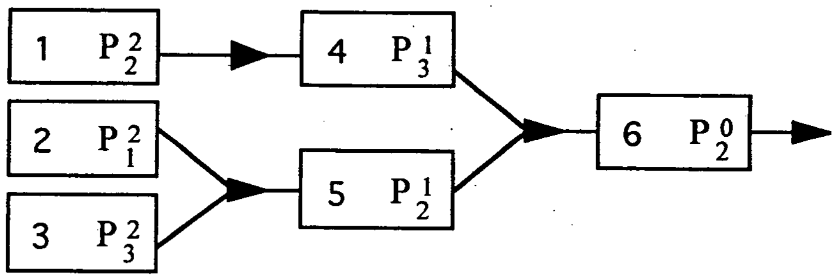


Figure 4. An assembly process and the required pools

The parameter δ is used to control the WIP. It is expected that the lower δ, the lower the production cycle, but the lower the utilization ratio of the machines. The goal of the following numerical examples is to test this assumption.

To perform these tests, we generate at random and in sequence assembly processes, and we schedule the operations as soon as the assembly processes are generated. This is done for different values of δ. For each value δ, we compute the utilization ratio (URBP) of the bottleneck pool of machines. This ratio reflects the productivity of the system. We also compute the mean value of the production cycle time (MPCT), as well as the mean utilization ratio of the storage facilities (URST). There are six operation stages in the example ($K = 5$). Table 3 gives the number of pools for each stage.

Table 3 - Number of pools per stage

Stage	K = 5	4	3	2	1	0
Number of pools	6	2	5	9	5	2

Each pool is made with 3 identical machines, and 4 storage identities are associated with each pool. For each manufacturing or assembly operation, the processing time is generated at random between 0 and 10. The assembly process is also generated at random, starting from level 0. The number of predecessors of each operation is bounded, as well as the total number of operations in a manufacturing process. As a consequence, some of the manufacturing processes have less than six operation stages.

We generate 1000 manufacturing processes, launched in production in the order they have been generated. For each value of δ, ($\delta = 0, 1, \ldots, 10$), we provide :

- The mean value of the production cycle time (MPCT). This value indicates, on the average, the tendency of the cycle times,
- The mean utilization ratio of the storage facilities (URST),
- The utilization ratio of the bottleneck pool (URBP).

The results are given in Table 4. The conclusion is straight foreward : the greater δ, the greater the productivity (since the value of the URBP increases with δ), but the greater the level of the WIP (since the URST increases with δ), and the greater the cycle time.

Table 4 - Control of the WIP

δ	MPCT	URST	URBP
0.	22.03	0.000	0.722
1.	24.67	0.050	0.776
2.	27.34	0.104	0.818
3.	30.23	0.162	0.858
4.	33.03	0.219	0.884
5.	35.96	0.276	0.906
6.	38.80	0.331	0.923
7.	41.51	0.379	0.931
8.	44.08	0.422	0.944
9.	46.37	0.460	0.950
10.	48.45	0.490	0.953

6. CONCLUSION

The algorithms presented in this paper are real time scheduling algorithms which allow to schedule new assembly process as soon as possible. As a consequence, the first idle periods used are the ones which are the closest to the current time. This guarantees a good use of the resource. Furthermore, by controling the flexibility of the system (i.e. the δ_i values), it is possible to reduce the WIP at the expense of the use of resources. In particular, as showed in the example presented in section 5, it is possible to adjust the WIP according to the required productivity.

REFERENCES

CALLAHAN J.R., "*The Nothing Hot Delay Problems in the Production of Steel*", PhD. Thesis, Department of Industrial Engineering, University of Toronto, Canada, 1971

CHAUVET F., LEVNER E., MEYZIN L.K., PROTH J.M., "*On-line Part Scheduling in a Surface Treatment System*", INRIA research reports, 1997, N. 3318, INRIA, Le Chesnay, France

CHU C., PROTH J.M., WANG L., "*Improving job-shops schedules through critical pairwise exchanges*", International Journal of Production Research, 1998, V. 36, N. 3, pp. 638-694

HALL N.G., SRISKANDARAJAH C., "*A survey of machine scheduling problems with blocking and no-wait in process*", Operations Research, 1996, V. 44, pp. 510-525

MC CORMICK S.T., PINEDO M.L., SHENKER S., WOLF B., "*Sequencing in an Assembly Line with Blocking to Minimize Cycle Time*", Operations Research, 1989, V. 37, pp. 925-935

RACHAMADUGU R. and STECKE K., "*Classification and review of FMS scheduling procedures*", Production Planning and Control, 1994, V. 5, N. 1, pp. 2-20

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, B.P. 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399

RR- 3432