No-wait scheduling in supply chain environment

Satyaveer Singh Chauhan, Valery Gordon, Jean-Marie Proth

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

No-wait scheduling in supply chain environment

S.S. Chauhan - V. Gordon - J.M. Proth

N° 4467
Mai 2002

THÈME

No-wait scheduling in supply chain environment

CHAUHAN[1] Satyaveer S, GORDON[2] Valery, and PROTH[3] Jean-Marie
[1,3] INRIA-SAGEP, Université de Metz, Ile du Saulcy, Metz, France.
[2]Institute of Engineering Cybernetics, National academy of sciences, Minsk, Belarus.

Abstract:

This paper presents an approach to schedule a project online in a supply chain without rescheduling or disturbing the previous schedules. The information at hand are the project requirements and the resources availability i.e. the busy status of resources. The objective is to find the shortest project completion time while following the no-wait strategy. Three algorithms are proposed. The first algorithm schedules the jobs online optimally for production processes that do not include assembly operations. The second algorithm utilizes the functionality of the first algorithm for scheduling the ordinary assembly processes (single assembly operation at the end of the processes). The third algorithm extends the approach of ordinary assembly process to schedule the complex assembly processes. A complex assembly process contains one assembly operation (if it is not the last one) or more than one assembly operations.

Key words: No-wait scheduling, Assembly, Supply chain, Online scheduling

[1] chauhan@loria.fr (Corresponding author)
[2] gordon@newman.bas-net.by
[3] proth@loria.fr

Ordonnancement immédiat dans un environnement de chaîne d'approvisionnement

CHAUHAN Satyaveer S[1], GORDON Valery[2], et PROTH Jean-Marie[3]
[1,3] INRIA-SAGEP, Université de Metz, Ile du Saulcy, Metz, France.
[2] Institut d'ingénierie Cybernétique, Académie Nationale de sciences, Minsk, Biélorussie.

Résumé:
Ce papier propose une approche pour l'ordonnancement en ligne d'un nouveau projet dans une chaîne d'approvisionnement, sans avoir à réordonnancer ni à remettre en question l'ordonnancement antérieur. Les informations disponibles sont les besoins du projet et la disponibilité des ressources i.e. les périodes d'utilisation des ressources. L'objectif est de trouver le temps d'achèvement le plus court du projet en appliquant la stratégie de non-attente. Trois algorithmes sont proposés : le premier algorithme ordonnance optimalement et en temps réel les tâches pour les processus de production qui n'incluent pas d'opération d'assemblage. Le second algorithme utilise les fonctionnalités du premier pour ordonnancer les processus d'assemblage ordinaires (une unique opération d'assemblage a lieu en fin de processus) . Le troisième algorithme étend l'approche du processus d'assemblage ordinaire afin d'ordonnancer des processus d'assemblage complexes. Un processus d'assemblage complexe contient soit une opération d'assemblage (à condition que cette opération ne soit pas la dernière) soit plusieurs opérations d'assemblage.

Mots clefs: Ordonnancement immédiat, assemblage, chaîne d'approvisionnement, ordonnancement temps-réel.

1. Introduction

During the past few years, the world's leading firms realized that formal relationships with suppliers and distributors, inflexible production systems, inefficient information technology (IT) systems, are the main causes of lack of competitiveness. The new trends in IT, the ever changing taste of customers, as well as quality and price pressure compel the firm to desert the traditional department organization and work in a supply chain framework, that is to work in an integration of organizations that cooperate for improving the flow of material and information from supplier to customers at lowest cost and highest speed.

The supply chain deals with projects and, at a given instant, several projects are under process. The goal is to schedule a new project utilizing idle periods of the resources at the best, while keeping the WIP as low as possible. The requirement of a new project can be anything from purchasing materials to delivery of finished goods and may require several resources such as machines, transportation systems, workers, space etc. One important requirement of supply chain is quick information flow, i.e. the system should be capable to gather information well in advance about the future status of all the available resources. In this paper, we assume that the management is informed about the busy status of these resources.

We consider a no-wait scheduling problem that consists in finding the lowest possible completion time of the project with assembly operations in a production system composed of several resources. Some or all of these resources are partially busy with other projects. We know the busy periods of these resources when a new project arrives in the system. The new project may utilize some or all of the resources in a specific given order. We assume that a resource can handle only one task at a time. The no-wait constraints mean that the next operation of the project must start as soon as the previous one is completed. The operation times vary between the minimum and the maximum permitted operation time. The possibility to extend the processing time gives little flexibility to the system. Some of the operations of the project are assembly operations. This means that several operations should be completed simultaneously to be able to start an assembly operation. The problem is to find, in real time, the schedule that leads to the lowest possible completion time of the project. The schedule we are looking for is a no-wait schedule and this constraint is crucial since it imposes the biggest difficulty in case of several assembly operations.

Let us review some results on no-wait scheduling problems. In the no-wait flow shop scheduling problem, n jobs are to be processed without interruption on m machines $M_1,...,M_m$ in this order such that each machine can only process one job at a time, and immediately after a job is finished on the machine M_k it has to be started on machine M_{k+1}. The objective is to minimize the makespan, that is, to find a schedule such that the last job to be processed on the last machine finishes as soon as possible. It is well known that this problem is *NP*-hard if m is a part of input (i.e. m is not fixed) [9], as well as when m is fixed and $m \geq 3$ [11]. For fixed $m=2$ this problem is solvable in $O(n \log n)$ time [13]; in this case the problem can be reformulated as a special case of the traveling salesman problem considered by Gilmore and Gomory [5]. The problem with m machines where the processing times are fixed on all except two machines can also be solved in polynomial time [17] as well as the problem for fixed $m \geq 4$ and unit-time jobs [3]. Kravchenko [6] proposes a polynomial time algorithm for the two-machine no-wait job shop problem of minimizing the total completion time where each job is a chain of unit processing time operations. If zero processing time

operations are allowed then the problem is *NP*-hard in the strong sense [6]. Two-machine job shop no-wait scheduling problem of minimizing makespan is NP-hard in the ordinary sense for unit processing time operations[16]. Kubiak [8] proposes a pseudo-polynomial algorithm for this problem. Sahni and Cho [15] prove strong NP-hardness of the no-wait open shop two-machine problem in which the second operation of each job must start immediately after the completion of its first operation. Röck[12] establishes that two-machine no-wait flow shop problem of minimizing maximum lateness and total completion time are strongly NP-hard. Agnetis [1] proposes an approach for the two and three machine no-wait flow shop problem with robotic cells. Lin and Cheng [10] consider no-wait two-machine flow shop in batching environment. Sriskandarajah [14] studies the no-wait flow shop problem in the case of parallel machines and proposes a heuristic algorithm with upper bound performance guarantee. Kumar *et al* [7] propose an approach for *m*-machines flow shop problem using genetic algorithms. A real time no-wait scheduling problem for surface treatment industry where one robot is used to transfer the jobs from one machine to another is considered by Chauvet *et al* [4]. An efficient algorithm with feasibility and optimality proofs is proposed for on-line scheduling. This approach is the most relevant to our problem since it considers the insertion of the new jobs into existing job processing (but unlike our problem, without assembly environment).

The rest of the paper is organized in 4 sections. In section 2, we describe an ordinary assembly process with problem formulation and present an approach to optimal scheduling. Section 3 presents a method to schedule a complex assembly process by extending the approach applied to the ordinary assembly process. Two numerical examples are presented in section 4 to illustrate the algorithm. Finally, the last section is the conclusion.

2. Ordinary assembly process

2.1 Definition:

An ordinary assembly process is a process that consists of several "Sequence of Operations" (SO) whose production is assembled by means of a single assembly operation. (See figure 1)

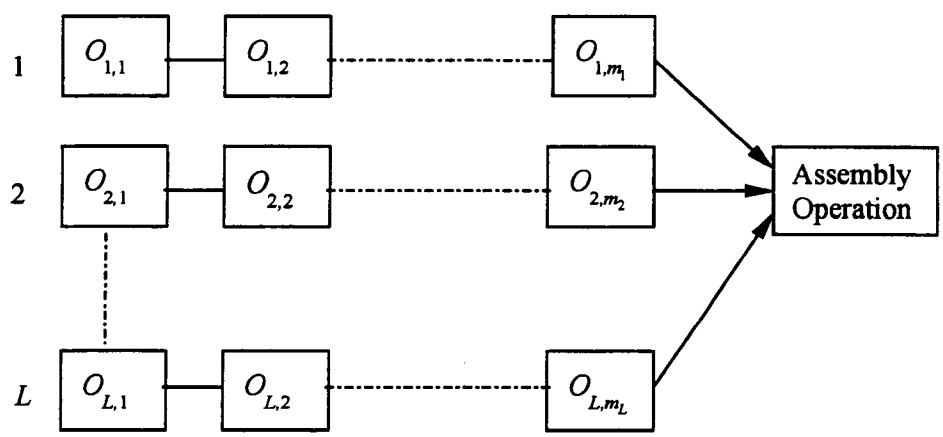


Fig. 1: An ordinary assembly process

In the above figure, each sequence of operations $l, 1 \le l \le L$, is composed of m_l operations, where l is the index of the *SO*.

2.2 Problem formulation:

Consider an ordinary assembly process, say A, which is composed of L sequences of operations and each sequence $l \in \{1,2,...,L\}$ is composed of m_l operations denoted by $O_{l,i}$, where $i=1,2,...,m_l$. The duration of operation $O_{l,i}$ can be chosen between $\theta_{l,i}$ and $\theta_{l,i}+\delta_{l,i}$, where $\theta_{l,i} \geq 0$ and $\delta_{l,i} \geq 0$. Similarly, the duration of the assembly operation can lie between θ_A and $\theta_A+\delta_A$.

Furthermore, several identical resources are available to perform the operation $O_{l,i}$ and this operation can be performed by any one of these available resources. The sets of resources dedicated to different operations are disjoint. Each resource has specific idle periods, also called available windows. Let $w_{l,i}$ denotes the total number of idle periods available for operation $O_{l,i}$. Preemption is not allowed, i.e. if operation starts in one window, it should be finished in the same window. In other words, the operation $O_{l,i}$ could be scheduled in only one window among the $w_{l,i}$ available windows. We assume that all the available windows concerning operation $O_{l,i}$ are numbered in the increasing order of their starting time, and if the starting times are the same, in the increasing order of their ending times. The starting and ending times of the $r_{l,i}$ th window of operation $O_{l,i}$ are denoted by $\alpha_{l,i,r_{l,i}}$ and $\beta_{l,i,r_{l,i}}$ respectively, where $r_{l,i} \in \{1,2,...,w_{l,i}\}$. We ignore all small windows that are not sufficient to perform the operation i.e. for all windows it is assumed that $\beta_{l,i,r_{l,i}} - \alpha_{l,i,r_{l,i}} \geq \theta_{l,i}$. We also assume that upper bound of the last window of each operation, including assembly operation, is infinite. It at least guarantees that a solution can be always achieved. The starting time and ending time of the assembly operation are denoted by α_{A,r_A} and β_{A,r_A} respectively, where $r_A=1,2,...,w_A$ and w_A is the number of idle windows available for assembly operation.

Let $x_{l,i}$ and $x_{l,i+1}$ denote the starting time and the ending time of operation $O_{l,i}$, which are to be found. Remind that operation $O_{l,i+1}$ should start immediately after $O_{l,i}$ finished, it means $x_{l,i+1}$ is the completion time of operation $O_{l,i}$ and the starting time of $O_{l,i+1}$. Now, the problem at hand is to find $x_{l,i}$ and $r_{l,i}$ for each i,l so that to minimize C, defined below in (4), subject to:

$$\theta_{l,i} \leq x_{l,i+1} - x_{l,i} \leq \theta_{l,i} + \delta_{l,i} \qquad (1)$$

$$x_{l,i} \geq \alpha_{l,i,r_{l,i}} \qquad (2)$$

$$x_{l,i+1} \leq \beta_{l,i,r_{l,i}} \qquad (3)$$

where $i=1,2,...,m_l$, $l=1,2,...,L$, and $r_{l,i} \in \{1,2,...,w_{l,i}\}$

$$x_{l,m_l+1} = C \qquad l=1,2,...,L \qquad (4)$$

$$C + \theta_A \leq \beta_{A,r_A} \qquad (5)$$

$$C \geq \alpha_{A,r_A} \qquad r_A \in \{1,2,...,w_A\} \qquad (6)$$

The above formulation expresses that:

- The objective is to reach the same completion time for each linear process, and this completion time should be minimum. (Constraint 4)
- An operation should be performed in an available window. (Constraints 2, 3)
- The processing time of each operation should be greater than or equal to the minimum required processing time and less than or equal to the maximum permitted processing time. (Constraint 1)

2.3. Algorithm:

In this section, we present an algorithm that solves the above problem. We first give a brief explanation of the algorithm and then present it in detail.

(i) Select the first feasible window where assembly operation could start.
We take the summation of the minimum processing time of all operations in each SO and then select the maximum one. We call this selected summation "earliest starting limit" or *ESL* for short. Now, we select a window among the available windows of the assembly operation, such that *ESL* should lie within the lower and upper bound of the window if possible. In this case, we replace the lower bound of the window by *ESL* and select the window. Otherwise, we select a first window whose lower bound is greater than or equal to the *ESL*.

(ii) Schedule the SO 1 such that the completion time of its last operation is greater than or equal to the lower bound of the window selected in step (i).
In the next section, we propose an algorithm that can schedule the operations according to (ii) and provide the minimum makespan. We call it "Early Start Time" of assembly and, in short, denote this time by *est*.

(iii) Find the maximum completion time of SO 1 without changing the assigned windows obtained in step (ii)
The algorithm for maximizing makespan within the given set of windows is also proposed in the next section. We call it "Latest Start Time" of assembly operation and in short, denote this time by *lst*. Thus, the SO 1 can be completed at any time in interval $\lfloor est_1, lst_1 \rfloor$. Set $\overset{*}{I} = \lfloor est_1, lst_1 \rfloor$

(iv) Schedule the next SO using an approach similar to step (ii) & (iii) and look if all the previously scheduled SO have a common set of completion time with this SO. If not, find the new feasible window for assembly operation and go to step (i).
For $l=2,...,L$:

Apply step (ii) & (iii) to the SO l. We obtain an interval $I_l = \lfloor est_l, lst_l \rfloor$ where the SO l can be completed.

Compute $\overset{*}{I} = \overset{*}{I} \cap I_l$. Two cases may happen:

a: $\overset{*}{I} \neq \phi$, In this case continue the computation with the next value of l.

b: $\overset{*}{I} = \phi$, we conclude that it is impossible to start the assembly operation before est_l. Thus, we update the window of the assembly operation by replacing its lower bound by est_l and start again from (ii). The process of updating the window is same as in step (i) except that we use est_l instead of *ESL*.

The above four steps give the general idea behind the algorithm. We have seen that the algorithm requires three computations

- Computation of the feasible window for the assembly operation.
- Computation of the minimum makespan
- Computation of the maximum makespan

The algorithm for computing the minimum makespan (*LCT*) is presented in section 2.4 and the algorithm for computing the maximum makespan (*MCT*) is presented in section 2.5 Finally, section 2.6 presents the algorithm (*OAA*) for scheduling an ordinary assembly process. It also includes the computation of feasible window to start the assembly operation.

2.4 Algorithm *LCT* (*Lowest completion time*)

This algorithm schedules the operations of *SO* l, where $l \in \{1,2,...,L\}$, and gives the minimum makespan.

1. $r_{l,i}=1$ for $i=1,2,...,m_l$
2. $t_1=\alpha_{l,1,r_h}$
3. $t_i=max(\alpha_{l,i,r_i}, \theta_{l,i-1}+t_{i-1})$, $i=2,..m_l$
4. $t_{m_l+1}=\theta_{l,m_l}+t_{m_l}$
5. $x_{l,m_l+1}=t_{m_l+1}$
6. $x_{l,i}=\max(t_i, x_{l,i+1}-\theta_{l,i}-\delta_{l,i})$ *for* $i=m_l, m_l-1,..., 1$
7. If inequality $x_{l,i+1}<\beta_{l,ir_{li}}$ holds for all $i=1,2,...,m_l$. Stop.
8. *Else*, for each $x_{l,i+1}>\beta_{l,i,r_{li}}$, *where* $i=1,2,...,m_l$

   8.1 $r_{l,i}=r_{l,i}+1$
9. Go to 2.

The above algorithm leads to the minimum completion time (x_{l,m_l+1}) of the SO l. The makespan given by ***LCT*** is feasible and optimal. For the optimality proof of above algorithm refer to Chauvet , Proth *et al* [4]. The steps 2 to 8 takes $O(m)$ time where m denotes the total operations in SO. Those steps can be repeated maximum of $n-m+1$ times, where $n=\sum_{i=1}^{i=m} w_i$ denotes the maximum number of idle windows in *SO*. Therefore, the complexity of the algorithm is $O(n \times m)$.

2.5. Algorithm *MCT* (Maximum completion time)

The computation of the maximum completion time of a "sequence of operation" is necessary to adjust the completion time of the SO that provide the component of the assembly. The result 1 is the basis of the algorithm that leads to maximum makespan of a SO in the available windows given by *LCT*.

1. $x^*_{l,i}=x_{l,i}$ for $i=1,2,...,m_l+1$
2. $temp=\min_{i=1,2..m_l}(\beta_{l,i}-x^*_{l,i+1})$
3. $x^*_{l,k}=x^*_{l,k}+temp$ for $k=1,2,...,m_l+1$

   Step 3 postpones all the operations globally as much as possible.
4. For $k=1,2,..m_l$, if $x_{l,k+1}=\beta_{l,k}$ then set $i^*=k$

   i^* *contains the greatest element of T (See result 1)*

5. For $i=i^*+1,..m_l$

   5.1 $temp=min(\theta_{l,i}+\delta_{l,i}-x^+_{l,i+1}+x^+_{l,i}, \min_{k=i,i+1..m_l}(\beta_{l,k}-x^+_{l,k+1}))$

   5.2 $x^+_{l,k+1}=x^+_{l,k+1}+temp$ for $k=i,i+1,...,m_l$

   Step 5 adjust the schedule according to conditions (i) and (ii) of result 1

6. Stop.

In the above algorithm $x^+_{m_l+1}$ is the maximum completion time. The complexity of steps 1 to 4 is on the order of $O(m)$, where m is the total number of operations in a SO. Steps 5.1 and 5.2 will be computed a maximum of $\frac{m(m+1)}{2}$ times. Therefore, the complexity of the whole algorithm is $O(m^2)$.

Result 1:

The feasible schedule $\{x_i\}_{i=1,2,...,m+1}$ in a given set of windows corresponds to the maximum makespan if and only if the following two conditions hold:

(i) $T\neq\phi$, where, $T=\{i / x_{i+1}=\beta_i, i=1,2,...,m\}$ and,

(ii) $x_{j+1}-x_j=\theta_j+\delta_j$ for $\forall\ j=k+1,...,m$

where k is the biggest element of T and m is the total number of operations in the SO.

Proof:

a. The conditions are necessary:

Assume that $\{x_i\}_{i=1,2,..m+1}$ denotes the schedule corresponding to the maximum makespan and condition (i) does no hold i.e. $T=\phi$. Then, set $\psi=\min_{i=1,2,..m}(\beta_i-x_{i+1})>0$ (7)

We then consider a set $\{X_i\}_{i=1,2,..m+1}$ defined as follows:

$$X_i=x_i+\psi \text{ for } i=1,2,...,m+1.$$

According to (7), $\{X_i\}_{i=1,2,..m+1}$ is a feasible schedule. Furthermore, since $\psi>0$, $X_i>x_i$ for $i=1,2,..m+1$. And, in particular, $X_{m+1}>x_{m+1}$, which is contradictory to the fact that x_{m+1} is a maximum makespan. Thus, condition (i) holds.

Now, assume that condition (i) holds and that k is the greatest element of T, and condition (ii) does not hold. Since (i) hold $\beta_i-x_{i+1}>0$ for any $i\in\{k+1,...,m\}$.
Since (ii) does not hold, there exists at least one $i^*\in\{k+1,...,m\}$ such that:

$$\theta_{i^*}+\delta_{i^*}>(x_{i^*+1}-x_{i^*}), \quad (8)$$

and $$\Delta_{i^*}=\min_{j=i^*,...,m}(\beta_j-x_{j+1})>0 \quad (9)$$

From (8) and (9) we deduce,

$$\psi_{i^*}=min(\Delta_{i^*},\theta_{i^*}+\delta_{i^*}-(x_{i^*+1}-x_{i^*}))>0 \quad (10)$$

Now, consider the solution $\{X_j\}_{j=1,2,..,m+1}$ defined as follows:

$X_j = x_j$ for $j=1,2,...,i^*$

$X_j = x_j + \psi_{i^*}$ for $j=i^*+1,...,m+1$.

The new solution, $\{X_j\}_{j=1,2,...,m+1}$ is feasible according to (8) and (10) and $X_{m+1} > x_{m+1}$, since $\psi_{i^*} > 0$, which is contradictory to hypothesis. Thus, condition (ii) must hold. As a consequence, we can say that the above conditions are necessary for a schedule to provide the maximum makespan.

b. The conditions are sufficient:

Let $\{x_i\}_{i=1,2,...,m+1}$ be a feasible solution that verify (i) and (ii). We denote by k' the greatest element of T for this solution.

If $k=k'$ then the two solutions are the same.

Assume that $k'<k$. Then, according to condition (ii) for any $j>k'$ we have,

$$x'_{j+1} = \beta_{k'} + \sum_{i=k'+1}^{j} (\theta_i + \delta_i) < \beta_j \qquad (11)$$

Since $k'<k$, we have:

$$x_{k'+1} \le \beta_{k'} \qquad (12)$$

By adding the same positive element to both sides of (11), we obtain:

$$x_{k+1} \le \beta_{k'} + \sum_{i=k'+1}^{k} (\theta_i + \delta_i) \qquad (13)$$

From (11) and (13), we derive:

$$x_{k+1} \le x'_{k+1} < \beta_k \qquad (14)$$

But $x_{k+1} = \beta_k$ according to condition (ii). Thus, Equation (14) shows that we reach a contradictory situation. The same kind of contradiction happens if $k<k'$.

Finally, (i) and (ii) are necessary and sufficient conditions for a feasible schedule to lead to the maximum makespan within a given set of windows. This completes the proof.

Result 2 The schedule given by *MCT* verifies the result 1.

Proof:

The steps 2 and 3 of the algorithm lead to a solution such that there exists i^* that verifies:

$x^+_{l,j^*+1} = \beta_{l,j^*}$ i.e. T is not empty.

Furthermore, for each $i>i^*$, step 5.1 computes:

$$\Delta = \min_{i=i^*+1,..m_l} (\beta_{l,i} - x^+_{l,i+1}) > 0$$

Assume that $\Delta = \beta_{l,k} - x^+_{l,k+1}$, where $k \in \{i,...,m_l\}$. Then, for the operation i, two cases may happen:

(a) Either, $\theta_{l,i} + \delta_{l,i} - (x^+_{i+1} - x^+_i) > \Delta$

In this case step 5.2 shift the operation's completion time by Δ for operation i onwards.

Since, for the operation k, $\beta_{l,k} - x^+_{l,k+1} = \Delta$ (before shifting) therefore, after shifting it will become $x^+_{l,k+1} = \beta_{l,k}$. Since, $k \ge i$, thus according to condition (i) $i^* = k$.

(b) Or, $\theta_{l,i}+\delta_{l,i}-(x^+_{i+1}-x^+_i)<\Delta$

In this case step 5.2 shift the operation's completion time by $(\theta_{l,i}+\delta_{l,i}-(x^+_{i+1}-x^+_i))$ for all operations i onwards. It means for operation i,

$x^+_{i+1}=x^+_{i+1}+(\theta_{l,i}+\delta_{l,i}-x^+_{i+1}+x^+_i)$

or $x^+_{i+1}-x^+_i=\theta_{l,i}+\delta_{l,i}$, this equality shows that one more operation time is extended to its maximum.

Finally, in each iteration of *MCT*, either an element belonging to $\{i^*+1,...,m\}$ is added into T or the operation time of the element is extended to its maximum. Hence, we can say that the schedule proposed by *MCT* verifies the result 1. This completes the proof.

2.6. Algorithm *OAA* (Ordinary assembly algorithm)

This algorithm utilizes the *LCT* for solving ordinary assembly process. To utilize the algorithm, we extend each *SO* by including assembly operation as the last operation (*$(m_l+1)th$* operation). We will hereafter refer this extended *SO* as assembly sequence (AS) i.e. *SO* associated with assembly operation. The explanations of the algorithm are already presented in section 2.3.

A. **Computation of the first feasible window where assembly operation could be performed.**

1. Set $r_{l,i}=1, i=1,2,...,m_l$, $l=1,2,...,L$
2. For $l=1,2,...,L$
   2.1. $h_l=\alpha_{l,1,r_{l1}}$
   2.2. $h_l=h_l+\theta_{li}$ for $i=1,2,...,m_l$
3. $T=\max_{l=1,2,..L}(h_l)$
4. Set $r_{l,m_l+1}=z$ for $l=1,2,...,L$, where z is such that:

   $\alpha_{l,m_l+1,z}\le T\le\beta_{l,m_l+1,z}$, $z\in(1,2,...,Z)$, where Z is the number of idle windows.

   m_l+1 denotes the assembly operation, which is same for all the "sequences of operations" (SO). Remember that, SO contain m_l operations only.

B. **Computation of minimum assembly starting time**

5. Set $f_1=0$ and $f_2=\infty$, where f_1, f_2 are the early and latest starting time of assembly operation.
6. Set $l=1$
7. Set $r_{l,m_l+1}=z$ and $r_{l,i}=1$ for $i=1,2,...,m_l$
8. Compute x_{l,m_l+1}, the earliest starting time of the assembly operation in AS l, using *LCT*.
9. Compute x^+_{l,m_l+1}, latest starting time for AS l, using *MCT*.

   x^+_{l,m_l+1} gives the latest starting time of assembly operation within the windows given by LMS i.e. the difference of x^+_{l,m_l+1} and x_{m_l+1} gives the maximum

flexibility to delay the assembly operation, by manipulating the starting and ending time of other operations, without changing the windows selected by LMS.

10. $z=r_{l,m_l+1}$
11. If $x_{m_l+1}>\alpha_{l,m_l+1,z}$ then set $\alpha_{l,m_l+1,z}=x_{m_l+1}$

    Since the last operation of AS is the assembly operation, the values $\alpha_{l,m_l+1,z}$ *(resp.* $\beta_{l,m_l+1,z}$*) are the same whatever* $l\in\{1,2,...,L\}$
12. If $f_1 < x_{m_l+1}$ then set $f_1 = x_{m_l+1}$
13. If $f_2 > x^+_{l,m_l+1}$ then set $f_2 = x^+_{l,m_l+1}$
14. If $f_1 > f_2$ go to 5
15. $l=l+1$
16. If $l>L$ then go to 19
17. $\alpha_{l,m_l+1,z}=f_1$
18. Go to 7
19. The value f_1 gives the optimal starting time of assembly. Stop.

In the above algorithm, f_1 gives the *est* (early start time) of the assembly operation and f_2 gives the *lst* (latest start time) of the assembly operation. Steps 5 to 18 can be repeated maximum of N times where $N=\sum_{l=1}^{L}\sum_{i=1}^{m_l} w_{l,i}$ is the total number of idle windows. Algorithm OAA uses *LCT* and *MCT* in step 8 and 9 respectively, so the complexity of these steps is the summation of the complexity of both algorithm, that is $O(m\times n+m^2)\cong O(m\times n)$. Thus the overall complexity of algorithms OAA is on the order of $O(N^2\times M)$, where $M=\sum_{l=1}^{L} m_l$ denotes total number of operations in assembly process and N is the total number of windows.

Finally, the last remaining step is to modify the $x_{l,i}$ values because f_1 is the minimum starting time and lie between the x^+_{l,m_l+1} and x_{l,m_l+1} for SO l. Modifying the $x_{l,i}$ values, assuming f_1 is the starting time of assembly, is straightforward and can be done by the following algorithm.

If $x_{l,m_l+1}\le f_1 \le x^+_{l,m_l+1}$, then compute λ_l such that,

$$\lambda_l=\frac{f_1-x_{l,m_l+1}}{x^+_{l,m_l+1}-x_{l,m_l+1}}$$

For $i=1,2,..m_l+1$

$x_{l,i}=x_{l,i}+\lambda_l*\left(x^+_{l,i}-x_{l,i}\right)$ where $l\in\{1,2,...,L\}$

Result 3
The first feasible solution obtained by *OAA* is optimal.

Proof:

We start with the earliest possible interval for the assembly operation, say $\left[\alpha_{A,r_A},\beta_{A,r_A}\right]$. We compute the minimal and maximum starting times of the assembly operation utilizing *LCA* and *MCA* for the first assembly sequence with $r_{1,i}=1$ for $i=1,2,..m_1$ and $r_{1,m_1+1}=r_A$, where r_A is the rank of $\left[\alpha_{A,r_A},\beta_{A,r_A}\right]$. We obtain a solution $I_1=[s_1^1,s_2^1]$, where s_1^1 and s_2^1 are the minimum and the maximum starting times of the assembly operation. The lower bound of I_1 denotes the minimum possible starting time of assembly operation if only the first AS is concerned.

Set: $\overset{*}{I}_1=I_1$

$r_A=r_{1,m_1+1}$.

Now, consider the second AS and compute the $I_2=[s_1^2,s_2^2]$ utilizing *LCT* and *MCT* with $r_{2,i}=1$ for $i=1,2,..m_2$ and $r_{2,m_2+1}=r_A$ since the assembly operation cannot start before r_A. If the new solution is such that $r_{2,m_2+1}>r_A$ then the solution obtained for first assembly sequence is invalid as it lies in window r_A. As a consequence we can say that there is no feasible solution for the assembly process before the window r_{2,m_2+1}. Thus, we set $r_A=r_{2,m_2+1}$ and restart the computation for the first assembly sequence as explained above.

If $r_{2,m_2+1}=r_A$, then the two following cases are possible:

a. $\overset{*}{I}_2=I_2\cap\overset{*}{I}_1\neq\phi$

Then the lower bound and the upper bound of $\overset{*}{I}_2$ denote the earliest start time and the latest start time of the assembly operation that concern AS 1 and 2. No other intervals that precedes the $\overset{*}{I}_1$ can be a solution.

b. $\overset{*}{I}_2=I_2\cap\overset{*}{I}_1=\phi$

In this case there are no solutions before $z=max(s_1^1,s_1^2)$. We thus restart the computation from first assembly sequence by modifying the lower bound of r_A by z i.e. $\alpha_{r_A}=z$.

Note: In the algorithm, we modify the lower bound of the *assembly operation window* in each iteration. When the schedule of first assembly sequence is completed, we set $\alpha_{r_A}=s_1^1$. At the second step $s_1^2\geq s_1^1$ and we set $\alpha_{r_A}=s_1^2$. It reduces the computational burden by avoiding the computation that has been already done.

We finally obtain an interval $\overset{*}{I}_2$ that is not empty. Now, we continue the same process to reach $\overset{*}{I}_3\neq\phi$, $\overset{*}{I}_4\neq\phi$,.. $\overset{*}{I}_L\neq\phi$. $\overset{*}{I}_l\neq\phi$ for $l\in\{1,2,...,L\}$ is the earliest interval when the assembly operation can start since, to schedule the AS l we start by assigning the earliest possible window to the assembly operation that is also feasible for the $l-1$ first assembly lines. Finally, the lower bound of $\overset{*}{I}_L$ gives the optimal starting time of assembly operation. This completes the proof.

3. Complex assembly process:

A complex assembly process is a process that contains one or more assembly operations connected with each other, directly or indirectly, by means of other ordinary operations. In the case of one assembly operation, the assembly operation must be followed by at least one operation. Figure 2 shows a typical complex assembly process.

The scheduling of a complex assembly process involves two main steps:

- Decomposition of complex assembly process into several ordinary assembly processes.
- Scheduling and coordination of all ordinary assembly processes.

In the following section we explain the decomposition method and the scheduling method for complex assembly process. The constraints of no-wait schedule and the variable operation times are also applied for complex assembly process.

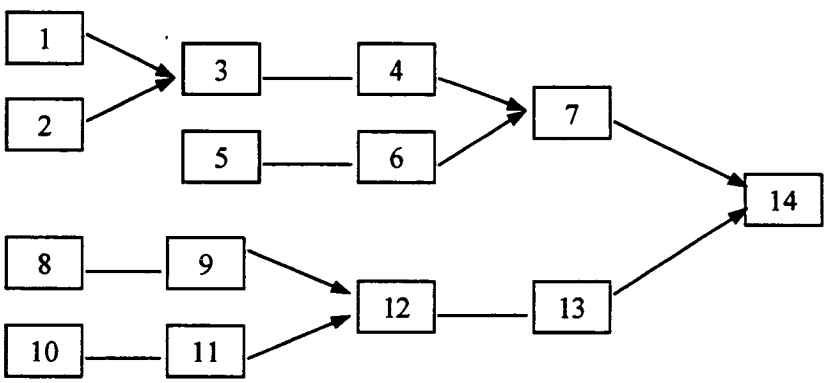


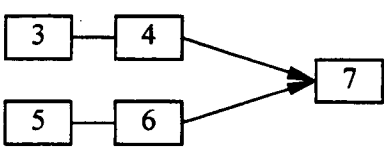
Figure 2: A complex assembly process

3.1 Decomposition

We use following three steps to decompose the complex assembly process.

1. We select an operation that has no successor. This operation is a root of the tree.
2. Then, we follow the sequence of predecessors until we find another assembly operation or an operation with no predecessor. Doing so, we obtain an ordinary assembly process that is a component of the complex assembly process.
3. If a leave of the component, obtained in step 2, is an assembly operation, then it also remains in the set of operations that have not been decomposed yet. So, such an operation is a leave for a component and a root for the remaining set of operations. If the set of remaining operation is not empty, we go to step 2, otherwise decomposition is finished.

Finally, we obtain several ordinary assembly processes (the components) when the decomposition ends. The components are ordered in sequence they are obtained.



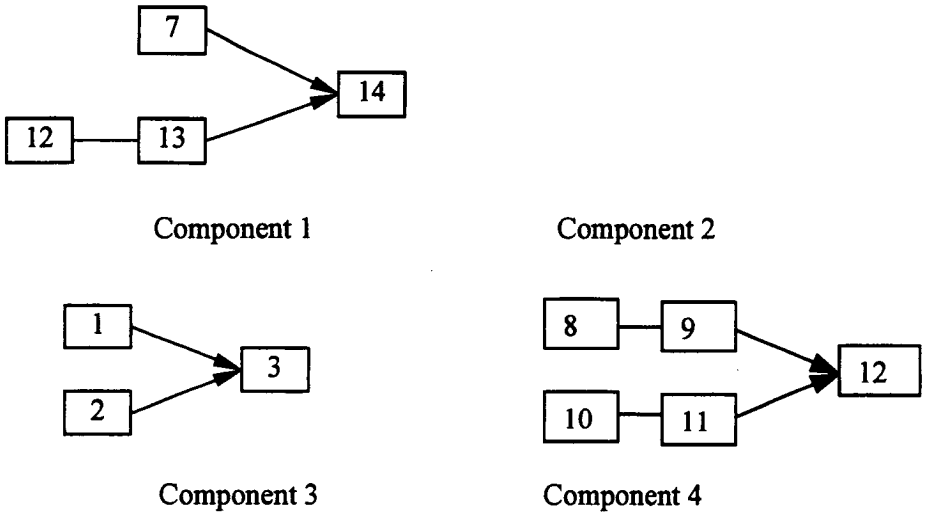


Figure 3: Elements of complex assembly process

The above steps are illustrated by the following example.

Example:

Consider the complex assembly process presented in figure 2. We can see that the operation 14 is the only operation with no successor. We select the operation 14 and go through its predecessors until we encounter an assembly operation or an operation with no predecessor. Doing so, we obtain the leaves 7 and 12 (assembly operations) of component 1. We have a choice to select any one between root 7 and 12 for decomposition. We select 7 and obtain component 2. Similarly, we can find component 3 after decomposition of component 2 and the remaining is component 4. Consequently, we obtain four ordinary assembly processes shown in figure 3.

Result 4: If $\{x_{l,i}\}_{i=1,2,..m_l+1}$ denotes the feasible schedule for an assembly sequence and T is the set of operations such that $T=\{i / x_{l,i+1}=\beta_i, i=1,2,..m_l\}$ then, for the set of available windows used, $x_{l,1}$ is the latest start time of the assembly sequence l, if and only if:

$x_{l,i+1}-x_{l,i}=\theta_{l,i}\ \forall\ i=1,2..k$.

where, $k=\begin{cases} \min_{i\in T}(i) & \text{if } T\neq\phi \\ m_l & \text{if } T=\phi \end{cases}$

Proof:

Assume $\{x_{l,i}\}_{i=1,2,..m_l+1}$ do not verify the above condition for $i=k$, then:

Set $\Delta=\min_{i=1,2,..k-1}(\beta_{l,i}-x_{l,i+1})>0$

Set $\psi=min(x_{l,k+1}-x_{l,k}-\theta_{l,k},\Delta)>0$

Now, compute $x_{l,i}=x_{l,i}+\psi$ for $i=1,2,..k$, this leads us to a new $x_{l,1}$ which is less than the previous one. This completes the proof.

3.2 Latest start time (*LST*)

The latest start time of assembly sequence l, for the same make span, is given by following algorithm. Let $z_{l,i}$ be the operation time of the i^{th} operation in assembly sequence l i.e. $z_{l,i}=x_{l,i+1}-x_{l,i}$

1. Compute $x_{l,1}=x_{l,2}-\theta_{l,1}$
2. Compute $Maxshift=\beta_{l,1}-x_{l,2}$
3. For $i=2,3..m_l$
    3.1 Compute $temp=min(z_{l,i}-\theta_{l,i},MaxShift)$
    3.2 Compute $x_{l,j}=x_{l,j}+temp$ for $j=1,..i$
    3.3 Compute $MaxShift=min(MaxShift-temp,\beta_{l,i}-x_{l,i+1})$
4. Stop.

The algorithm verifies the result 4. Step 1 adjusts the duration of first operation to the minimum required. Step 2 looks the maximum possibility of postponing the operation's starting time and completion time. Step 3 checks each operation duration and reduces it as much as possible. The process will continue until it is stopped by condition of result 4 i.e. when any one of the operation's completion time reach the upper bound of the corresponding window or the operation duration of all operation is minimum. The complexity of *LST* is $O(m^2)$, where m is the total operations in SO.

Applying *LCT* on the schedule given by *MCT* gives the latest start time and latest completion time of each operation. Hereafter, using *MCT* means using *MCT* and *LST* both.

3.3 Algorithm for Complex assembly process

In this section we will first explain our approach to schedule complex assembly process with the help of the example shown above, and then we present the algorithm.

Consider the components of the complex assembly process shown in figure 2 and whose components are presented in figure 3. We first consider component 1 and then the other components in the order they are obtained. We use the ordinary assembly algorithm (*OAA*) to schedule the operations of component 1. We know that the *OAA* starts by defining the first feasible window for assembly operation. To define this window, we sum up the minimum processing time of each SO from the beginning of the leaf to the beginning of the assembly operation (root) under consideration and select the maximum value for defining the first feasible window of assembly operation. In our case the assembly operation is 14 and the sequences are (1,3,4,7), (2,3,4,7), (5,6,7), (8,9,12,13), and (10,11,12,13).

By applying *OAA* to schedule component 1, we obtain est_7, lst_7, est_{12} and lst_{12} that is the earliest start time and the latest start time of operation 7 and 12 respectively. est_7 and est_{12} shows that operation 7 and 12 can not start before est_7 , est_{12} respectively. In the computation of these times we didn't consider the constraints derived from the components connected to 7 and 12. In the next steps, we check if these times are feasible to start assembly operation 7 and 12 (i.e. component 2 and component 4) or not. If not we will shift the starting time of operation 14 accordingly.

est_7 is the earliest start time for assembly operation 7, so we ignore all the idle windows of 7 whose upper bound is less than est_7 .Now, we schedule the second ordinary assembly process, component 2, as above using the same algorithm (*OAA*). Assume that the algorithm gives est_3 and lst_3 for assembly operation 3. The operation 3 is an assembly operation, so we schedule the component corresponding to operation 3 i.e. component 3 after updating its idle

windows such that first available window's upper limit should be greater than est_3. Applying *OAA* on component 3 gives est_3' and lst_3', the earliest start time and latest start time of assembly operation 3. Note that, $est_3' \geq est_3$. To carry out the verification, we look at the common interval, if any, between $\{est_3, lst_3\}$ and $\{est_3', lst_3'\}$. Then:

- If there is a common interval, we update the corresponding x_{ij} values (for component 2 and component 3) according to this common interval. We move to the next unprocessed assembly operation. In our case it is component 4. We schedule the operations of element 4 using the same procedure.
- If there are no common intervals, we update the available window of assembly operation 3 such that the starting time of the first available window of assembly operation 3 is either greater than or equal to est_3'. We schedule again the successor element of component 3 i.e. component 2 and repeat the above process.

Note: If we find the common interval for operation 3, then we update the x_{ij} values of component 3 and component 2 accordingly. Assume it gives est_7' and lst_7'. To carry out the verification, we look at the common intervals, if any, between $\{est_7, lst_7\}$ and $\{est_7', lst_7'\}$. The rest of the process is the same.

The above procedure is repeated until we find a common interval for all the nodes of complex assembly process. Finally, we select the lower bound of the common interval of operation 14. It is the time when assembly operation 14 can start. The makespan of the whole process is the summation of the starting time of operation 14 and the minimum operation time of 14.
Based on the above idea we can summarize the steps as follows:

Step1: Find the first possible window where the assembly operation can start considering all its predecessors.
Step2: Decompose the complex assembly process into ordinary assembly processes.
Step3: Schedule the first component using *OAA* and store the early start time and latest start time of all SO.
Step4: Check if the leave of the ordinary assembly process solved in step 2 is an assembly operation. If yes, schedule the corresponding ordinary assembly process, next element, using *OAA* and note the *est* and *lst* of **assembly operation**. Verify, if *est* and *lst* obtained for **assembly operation** have an interval in common with the corresponding early start time and latest start time of the ***SO*** (computed in step 2*)*. If yes, update the schedule time according to the common interval and schedule the next element-using *OAA*; otherwise, schedule the **previous element** (successor assembly process) again with the updated window.
Step 5: If common intervals are obtained for all the nodes, we select the lower bound of common interval for the last operation and update all the values accordingly and stop the process.

The above algorithm mainly does three computations:

- Computing the minimal possible start time to define the window where assembly could start.
- Updating the windows, if the condition of common interval fails.
- Scheduling the ordinary assembly process to find the early start time and the latest start time.

In section 3.4, we present algorithm (*MST*) that develop in detail step 1 of the algorithm. Section 3.5 is devoted to algorithm UW that update the windows of operation in case the condition of common interval fail (step 5).

3.4 Algorithm *MST* (probable starting time of assembly)

1. $Max=0$
2. For $l=1,2...,L$
   2.1. If $O_{l,1}$ is assembly operation, then utilize *MST* to get estimated starting time of component corresponding to $O_{l,1}$ and set it into t, else set $t=\alpha_{l,1,1}$.
   2.2. Set $t=max(t+\theta_{l,i}, \alpha_{l,i,r_{l,i}})$ For $i=1,2,..m_l$.
   2.3. If $(t>Max)$ then $Max=t$.
3. Stop.

In the above algorithm Max returns the minimum possible starting time of specific assembly.

The complexity of *MST* is $O(M)$, where M is total number of operations in complex assembly process.

3.5 Algorithm *UW* (Updating windows)

This subroutine updates the available windows of $O_{l,i}$ such that the starting time of the first window is either *est* or greater than *est.*

1. Find k such that
   $\alpha_{l,i,k} \le est < \beta_{l,i,k}$ or $\alpha_{l,i,k}$ is the minimal lower bound that verify $est<\alpha_{l,i,k}$ where $k \in \{1,2,..w_{l,i}\}$ and $w_{l,i}$ is the maximum number of available windows.
2. If $\alpha_{l,i,k} \le est < \beta_{l,i,k}$, then set $\alpha_{l,i,k}=est$
3. Set $w_{l,i}=w_{l,i}-k+1$
4. Set $\alpha_{l,i,j}=\alpha_{l,i,j+k-1}$ and $\beta_{l,i,j}=\beta_{l,i,j+k-1}$, for $j=1,2,..w_{l,i}$

It is understandable that all available windows having a lower bounds and upper bounds less than *est* are useless. Further, if *est* lies between the lower bound and upper bound of first window, we replace the lower bound of window by *est*. The complexity of the above algorithm is $O(w)$, where w is the maximum number of idle windows.

3.6 The first feasible solution obtained by algorithm is optimal

The complex assembly algorithm is the extension of ordinary assembly algorithm. Initially, we considered sub-assembly as an ordinary operation and scheduled it using *OAA*. At the second stage, we only verify if the schedule is OK for sub assembly or not. If not, we remove the unfeasible windows and restart with the same process.

This algorithm utilizes *OAA* for scheduling each component of complex assembly process. The complexity of *OAA* is on the order of $O(N_i^2 \times M_i)$ where M_i and N_i respectively denotes the total number of operations and total number of idle windows in the i th element. Therefore the complexity of overall algorithm can be denoted by:

$O(\sum_{i=1}^{k} N_i^2 \times M_i)$, where k is the total elements in complex assembly process.

4. Numerical example:

This section presents two examples for the complex assembly process. Fig. 2 shows the layout of the complex assembly process considered for both examples. The layout shows that the complex assembly process is made up of 14 operations including 4 assembly operations. The minimum required operation time and permitted flexibility is shown in table 1. Fig.4 shows the available windows for each operations and the schedule provided by the algorithms. The dark bars represent the busy periods for corresponding operation and hatched bar shows the schedule obtained for the same operation. The busy periods for example 2 are same except for first operation where we blocked the first available window up to 11. Table 2 presents the new schedule.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
θ (Minimum)	2	3	5	1	2	5	8	3	3	3	1	5	5	2
δ (Flexibility)	3	4	7	5	2	4	4	3	5	4	2	5	6	4

Table 1 Operation time and flexibility of operation

Operation	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
Timings	45-47	40-47	47-52	52-53	45-47	47-53	53-61	35-41	41-49	45-48	48-49	49-54	54-61	61-63

Table 2 Schedule for example 2

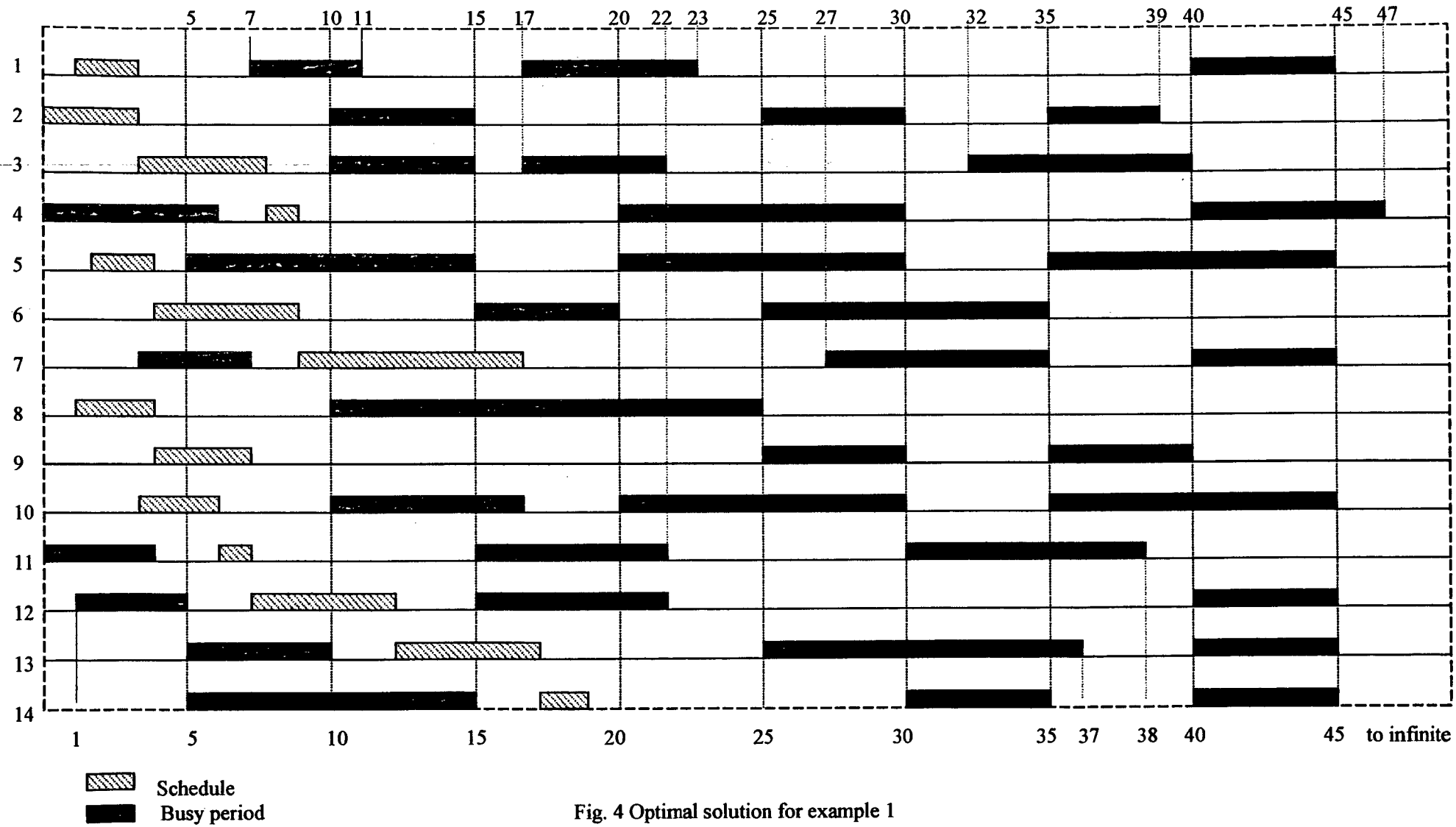


Fig. 4 Optimal solution for example 1

5. Conclusion:

This paper presents an approach to utilize the resources efficiently while minimizing the WIP as much as possible. The algorithms presented in this paper are real time algorithms. These algorithms intelligently search the solution right from the first idle periods until they locate the optimal schedule and intermittently remove the infeasible periods to improve the performance. The computation of two completion times for single schedule curtails the unnecessary computation of same schedule until the completion range suits to all *"sequences of operations"* of the same assembly process. Both algorithms provide optimal solution and promise to come up with a solution in case the upper bound of the last window of each operation is infinite. The performance in terms of the computation time deteriorates as the number of "sequences of operations" within an assembly operation and the number of assembly operations within a complex assembly process increase. The variable operation timing may seem impractical at the first instance, but it is often used in the heat treatment process of the chemical industry. These algorithms are equally useful for the resources with fixed operation timings and can be utilized by setting δ to zero value.

References

[1] Agnetis A., Scheduling no-wait robotic cells with two and three machines, European Journal of Operational Research 123 (2) (2000) pp. 303-314.

[2] Baker, K., Introduction of Sequencing and Scheduling, Wiley, New York, 1974.

[3] Blazewicz J., Kubiak W. and Szwarcfiter J. Scheduling unit-time tasks on flow-shops under resource constraints. Annals of Operations Research, 16, (1988), pp. 255-266.

[4] Chauvet F., Levner E., Meyzin L. K., Proth J-M, On-line scheduling in a surface treatment system, European Journal of Operational Research 120 (2)(2000) pp. 382-392.

[5] Gilmore P.C., and Gomory R.E., Sequencing a one-state variable machine: Solvable case of the traveling salesman problem. Operations. Research. 12, (1964), pp. 655-679.

[6] Kravchenko S.A., A polynomial algorithm for a two-machine no-wait job-shop scheduling problem, European Journal of Operational Research 106 (1) (1998) pp. 101-107.

[7] Kumar S., Bagchi T. P., Sriskandarajah C., Lot streaming and scheduling heuristics for m-machine no-wait flowshops, Computers and Industrial Engineering 2000 38 (1) pp. 149-172.

[8] Kubiak W., A pseudo-polynomial algorithm for a two-machine no-wait job shop scheduling problem. European Journal of Operations Research , 43, (1989), pp. 267-270.

[9] Lenstra J, K., Rinnooy Kan A.H.G., Brucker P., Complexity of machine scheduling problems. Annals. of Discrete Mathematics, 1,(1977), pp. 343-362.

[10] Lin B. M.T., Cheng T.C. Edwin, Batch scheduling in the no-wait two-machine flow shop to minimize the makespan, Computers and Operations Research 28 (7)(2001) pp. 613-624.

[11] Röck H., The three-machine no-wait flow shop is NP-complete. Journal of the Association for Computing Machinery. 31, (1984), pp. 336-345.

[12] Röck H., Some new results in flow shop scheduling. ZOR-Mathematical Methods of Operations Research, 28, (1984) pp. 1-16.

[13] Reddi S.S. and Ramamoorthy C.V., On the flowshop sequencing with no-wait in process. Journal of Operations Research Society. 23, (1972) pp. 323-331.

[14] Sriskandarajah C., Performance of scheduling algorithms for no-wait flowshops with parallel machines, European Journal of Operational Research, (1993) 70 (3) pp. 365-378.

[15] Sahni S. and Cho Y., Complexity of scheduling shops with no-wait in process. Mathematics of Operations Research. 4, (1979), pp.448-457.

[16] Timkovsky V.G. On the complexity of scheduling an arbitrary system. Soviet Journal of Computer and System science, 5, (1985), pp. 46-52.

[17] Van der Veen J. a.a. and Van Dal R., Solvable cases of the no-wait flow-shop scheduling problem. Journal Operations. Research. Society. 42, (1991), pp. 971-980.

Appendix

In this section we present the complex assembly algorithm in detail, easy to code in programming language. This algorithm starts with first component and schedule the other components recursively.

Algorithm *CSA* (Complex Assembly Algorithm)

1. Use *MST* to get the minimum possible starting time for assembly operation of this element (start with first).
2. Apply *OAA* and *MST* on this element (say it parent element).
   Step 2 gives early start time and latest start time of all operations. If this element contains L SO, then the est & lst of first operation of each SO can be denoted by est_l *&* lst_l *for* l=1,2,..L.
3. For l=1,2,..L
   - 3.1 If $O_{l,1}$ is not an assembly operation, go to 4
   - 3.2 Solve the element that contains $O_{l,1}$ as root using *CSA*. Say it child element. It will give est_l^* and lst_l^*, the early start time and latest start time of assembly operation $O_{l,1}$.
   - 3.3 Check if (est_l , lst_l) and (est_l^* , lst_l^*) have an interval in common. If yes, adjust the all operation timings (parent element & child element) according to common interval.
     For adjusting the operation timings, we presented a formula at the end of OAA.
   - 3.4 If not, update the $O_{l,1}$ *'s* windows using *UW* and est_l^*. Go to 1.
     In this case we schedule the parent assembly again with updated windows.
4. End of for loop.

Finally, we select the lower bound of common interval for first component. The total makespan is the summation of minimum operation time of assembly operation (root) of the first component and the lower bound of the common interval of this component.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399